Fachhochschule Köln
University of Applied Sciences Cologne
Abteilung Gummersbach
Fachbereich Informatik

Diplomarbeit
zur Erlangung
des Diplomgrades
Diplom-Informatiker (FH)
in der Fachrichtung Wirtschaftsinformatik

„**Simulation eines Mobilfunknetzes auf Basis von J2EE**“

Erstprüfer:	Prof. Dr. Heide Faeskorn-Woyke
Zweitprüfer:	Dipl. Ing. Ralf Kneemeyer
vorgelegt am:	15.01.2003
von:	Ramin Ziai
aus:	Paulistrasse 13
	50226 Frechen
E-Mail:	Ramin.Ziai@web.de
Matr.-Nr.:	10233536

Inhaltsverzeichnis

Abbildungsverzeichnis

Glossar

3rd Party Dritt-Hersteller

Applikationsserver Dient der Verwaltung und dem Betrieb von Applikationen oder Teilapplikationen. Application Server stellen Applikationsverfügbarkeit und -qualität sicher.

CalledParty Teilnehmer zu dem ein Call aufgebaut wird.

CallingParty Teilnehmer der einen Call aufbaut.

CSD – Circuit Switched Data. Trägerdienst im GSM-Netz.

CORBA – Common Object Request Broker Architecture. Technologie, welche eine orts-, plattform- und implementations-unabhängige Kommunikation zwischen Applikationen erlaubt.

DocumentHandler Programm zur Verarbeitung einer XML-Datei.

Enabling Produkt Vorprodukt welches Daten bereitstellt, die zu einem Produkt veredelt werden können.

Enterprise-Bean Serverseitige Komponente in einem mehrschichtigen System.

GPRS – Global Packet Radio System. Paketorientierter Trägerdienst im GSM Netz.

Hashtable Objekt in einer Programmiersprache zur temporären Speicherung von Daten.

HSCSD – Highspeed Circuit Switched Data. Trägerdienst im GSM-Netz.

IN – Intelligent Network. Netzwerkkonzept, bei dem die Vermittlungssteuerung vom technischen Vermittlungsvorgang getrennt ist.

J2EE – Java 2 Enterprise Edition. Standart zur Entwicklung von Multi-Tier Architekturen.

JMS – Java Messaging Service. Framework zur Entwicklung von Nachrichtendiensten.

JSP – Java Server Pages. Serverseitige Skripttechnologie.

JVM – Java Virtual Maschine. Laufzeitumgebung für Java-Programme.

Location Based Services Standortbezogene Dienste

Mobilstation Endgeräte für die mobile Kommunikation. Typischerweise ein Handy.

MOM – Message Orientated Middleware. Oberbegriff für die verteilte Kommunikation mit Nachrichten.

MSISDN – Mobile Station ISDN - internationale Rufnummer des Teilnehmers

Multi-Tier Mehrschichtiges System.

Non-Voice Dienste Datendienste.

Parser Programm zur Verarbeitung von XML-Dokumenten.

Pattern Programmiermuster für häufig wiederkehrende Probleme.

PDA – Personal Digital Assistent. Digitaler Organisator insbesondere zur Termin- und Aufgabenverwaltung.

Roaming. Ermöglicht es Benutzern von schnurlosen oder Mobiltelefonen zwischen den einzelnen Zellen in einem Mobilfunknetz zu wechseln, ohne dass momentan bestehende Verbindungen dabei abbrechen.

SAX – Simple API for XML. Programmiergerüst zur Entwicklung von XML-Anwendungen.

SMS – Short Message System. Trägerdienst zur Übermittlung von Textnachrichten.

SSL – Secure Socket Layer. Sicherheitsprotokoll für den Austausch von Daten im Internet.

Tier Schicht in einem mehrschichtigen System.

UMTS Universal Mobile Telecommunications System Die dritte Generation des Mobilfunks. Ermöglicht hohe Datenübertragungsraten von theoretisch 2 Mbit.

Value Added Service Provider Dritthersteller der aus Informationen einen Mehrwert schafft.

WAP – Wireless Application Protocol. Protokoll zur Darstellung von Inhalten aus dem WWW auf Geräten, die nur langsame Datenübertragung und eingeschränkte Darstellung erlauben.

Kapitel 1

Abstrakt

1.1 Motivation und Zielsetzung der Arbeit

In Zukunft sollen Non-Voice-Dienste verstärkt Traffic und Umsatz für Mobilfunknetzbetreiber generieren. Entscheidende Faktoren für den Erfolg sind dabei Personalisierung und damit auch standortbezogene Dienste. Die geschäftsentscheidenden Nutzerinformationen hierfür - Ort und Identität der Mobilfunkkunden - befinden sich in der Domäne der Netzbetreiber. Zur effektiven Umsetzung dieser Vermarktungspotentiale ist es zwingend erforderlich Dritt-Hersteller, sogenannte Value Added Service Provider, bei der Erschließung des neuen Marktsegmentes zu beteiligen.

In diesem heterogenen Markt, in dem eine Vielzahl unterschiedlicher Anwendungen denkbar sind, ist es die Aufgabe des Value Added Service Provider (VASP), Inhalte und Technologien zielgruppengerecht miteinander zu verknüpfen. Um den VASP zur Entwicklung entsprechender attraktiver Applikationen zu animieren, soll ihm eine Testplattform zur Verfügung gestellt.

Ziel der vorliegenden Arbeit ist es, eine solche Testumgebung in Form eines Netzsimulators zu entwickeln. Der Focus der Anwendung liegt dabei auf der Simulation der Schnittstellen und des Schnittstellenverhaltens der Vorprodukte (Enabling Produkte). Bei diesen Enabling Produkte handelt es sich um Dienste innerhalb des T-Mobile Datennetzes, die die zu vermarktenden Nutzerinformationen für den VASP bereitstellen.

Als Kerntechnologie für die Realisierung des Netzsimulators wird die Java 2 Enterprise Edition (J2EE) eingesetzt. Sie ermöglicht die Entwicklung einer plattformneutralen, komponentenbasierten und skalierbaren Anwendung.

1.2 Aufbau des Dokumentes

Im Kapitel „Vorbetrachtung“ erfolgt eine Beleuchtung der wirtschaftlichen Relevanz von „Non-Voice“-Diensten für ein Mobilfunkunternehmen. Weiterhin wird beschrieben wie sich aus diesem neuen Geschäftsfeld, das 3rd Party Enabling Programm entwickelt hat und welche Enabling Produkte ein besondere Rolle spielen. Das Kapitel „Situationsbeschreibung“ sondiert den gegenwärtigen Stand der Technik und bietet einen kurzen Einblick in die Architektur eines Mobilfunksystems. Daran anschließend wird in Kapitel 3 „Anwendungsdefinition“ besprochen, was der Netzsimulator leisten soll. Hierzu wird zunächst die Anwendungsidee vorgestellt, gefolgt von der Anforderungsspezifikation und den Anwendungsfällen. Das nächste Kapitel schildert dann sämtliche Technologien, die der Arbeit zugrunde liegen. Im Mittelpunkt des Interesses steht hier die J2EE-Technologie. Nachdem die wirtschaftlichen Aspekte beleuchtet und die technischen Grundlagen vermittelt wurden, widmet sich das sechste Kapitel der Umsetzung der zuvor beschriebenen Anwendungsidee. In dem Abschnitt werden die entwickelten Systemkomponenten vorgestellt und ausführlich beschrieben. Das Kapitel endet mit Implementierungshinweisen, die die Programmier- und Systemumgebung betreffen. Das letzte Kapitel „Ausblick“ widmet sich den wirtschaftlichen Perspektiven von Non-Voice Diensten sowie möglichen technischen Entwicklungsszenarien.

Kapitel 2

Vorbetrachtung

Einleitend erscheint eine Betrachtung von Non-Voice-Diensten und deren wirtschaftlicher Bedeutung notwendig. Anschließend wird das 3rd Party Enabling Programm von T-Mobile vorgestellt, in dessen Mittelpunkt die Entwicklungsförderung von Non-Voice-Diensten sowie die Vermarktung der Nutzerinformationen liegt. Den Enabling Produkten, in denen sich die Nutzerdaten befinden, widmet sich schließlich der letzte Abschnitt.

2.1 Non-Voice Dienste im Mobilfunk

Mobiltelekommunikation und Internet haben sich in den vergangenen Jahren zu den treibenden Kräften zur Steigerung der industriellen Produktivität entwickelt. Gleichzeitig hat es die rasante Weiterentwicklung der zugrundeliegenden Technologien ermöglicht, dass die Massenanwendung im privaten Bereich Einzug erhielten. Mit dem Zusammenwachsen dieser beiden Märkte, entstehen nun neue Geschäftsfelder, die in einer Vielzahl von Branchen als Innovationsmotor wirken und ein gesamtwirtschaftliches Potential entfalten können.

Ein Bereich, der von dieser Entwicklung stark profitiert, ist das Non-Voice-Geschäft. Immer mehr Menschen bedienen sich mittlerweile sogenannter Non-Voice-Services um informiert zu sein, zu kommunizieren oder sich unterhalten zu lassen. Die mobile Datenkommunikation umfasst dabei Informationen von Sport- oder Börsennachrichten bis hinzu herunterladbaren Bildern, Melodien, Stadtplänen und Wetterkarten. Der wohl bekannteste Non-Voice-Service ist der Short Message Service (SMS).

Mit der zunehmenden Bedeutung dieses Marktsegmentes, müssen die Mobilfunkbetreiber neue Einnahmequellen erschließen. Dabei stehen in der

Startphase noch nicht einmal die Erlöserwartungen des Non-Voice-Geschäftes im Zentrum der Überlegungen, sondern die Steigerung der Kundenloyalität zur Stabilisierung der Kundenbasis.

Nach dem Start der ersten GPRS-Dienste und vor allem mit der Inbetriebnahme der UMTS-Netze wird der Einfluss der Non-Voice-Umsätze auf die Ergebnissituation immer größer werden. Laut einer Studie von JP Morgan/ Arthur Andersen wird der Anteil der Non-Voice-Dienste den Anteil der reinen Sprachkommunikation bereits in wenigen Jahren übertreffen.

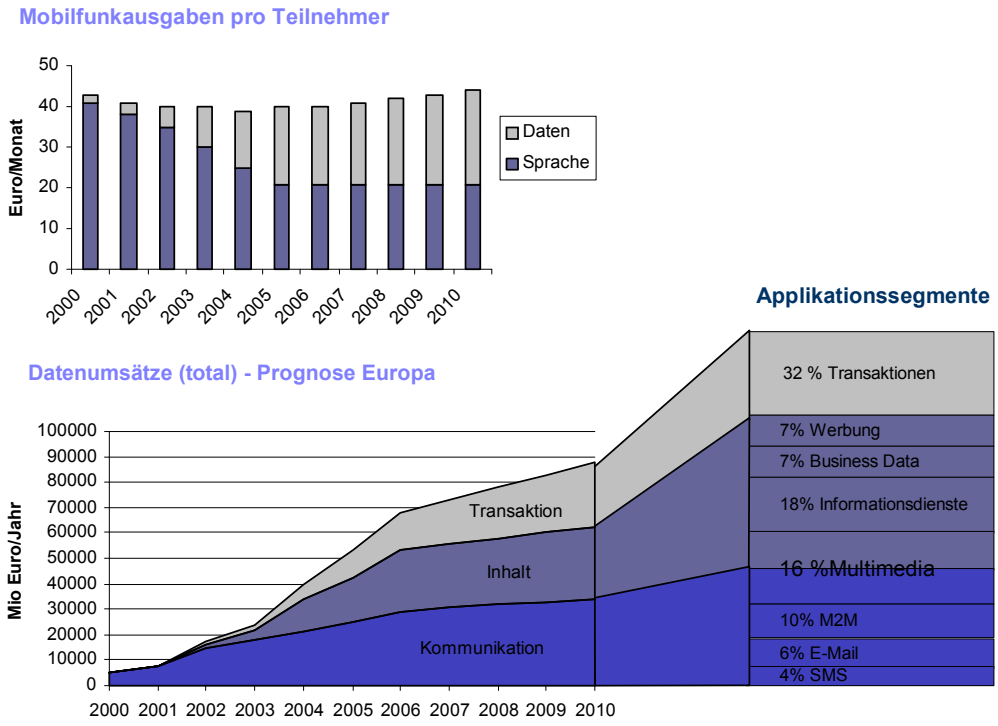

Abbildung 1 - Umsatzprognose für Non-Voice-Dienste

Andere Studien, beispielsweise von BBDO Consulting[1] und IZT[2], zeigen eine ähnliche Entwicklung. Die wichtigste aktuelle Herausforderung eines Mobilfunkunternehmens ist daher die Ausrichtung auf das Non-Voice- bzw. UMTS-Zeitalter.

[1] BBDO, Studie „MCommerce mit UMTS“, S. 7
[2] IZT, Studie „Entwicklung und zukünftige Bedeutung mobiler Multimediadienste“, S. 2

2.2 Das 3rd Party Enabling Programm[3]

Wie bereits im vorigen Abschnitt erwähnt, ist es von zentraler Bedeutung ein hohes Maß an Kundenvertrauen aufzubauen, das vor allem auf die persönliche Kommunikation als Schwerpunkt zielt. Ist dies einmal gelungen, kann eine Vermarktung der Nutzerinformationen erfolgen. Statt eigene Applikation zu entwickeln, die für Kunden attraktive Mehrwertdienste bereitstellen, ist es möglich externe Content- und Dienstanbieter (Value Added Service Provider) in die M-Commerce Aktivitäten mit einzubinden. Die Gründe hierfür sind:

- Trotz der enormen Bedeutung für den Erfolg des Non-Voice-Geschäftes, wird die Entwicklung und Realisierung von neuen Applikationen und Geschäftsmodellen, die nicht zum Kerngeschäft des Unternehmens gehören, meist ohne einen klar strukturierten Entwicklungsprozess - und zudem oft zögerlich - vorangetrieben.

- Das Non-Voice Geschäft ist durch eine komplexe und hoch differenzierte Wertschöpfungskette gekennzeichnet. Hinzu kommen eine überdurchschnittliche generelle Innovationsgeschwindigkeit sowie sehr kurze Technologiezyklen. Für die Zukunft wird erwartet, dass sich diese Trends weiter verstärken und insgesamt zu einem Bedarf an noch spezialisierterem Know-how und zu immer höheren Anforderungen an eine möglichst kurze „Time-to-Market“ führen.

Dies macht deutlich das die Value Added Service Provider bei der Umgewichtung des Sprachverkehrs hin zum Non-Voice in ihrer Bedeutung stark aufgewertet werden. Sie werden dabei völlig neue Vermarktungskanäle erschließen und so neue Umsätze generieren können. Durch die Lokalisierbarkeit[4] können für die Kunden innovative Services geschaffen werden, die in der Lage sind, die persönliche Effizienz zu steigern.

---

[3] siehe http://www.t-d1-developercenter.de/content/competence/specification.shtml
[4] Lokalisierbarkeit = Ortung von Mobilfunkkunden bzw. Mobilgeräten. Siehe auch Kapitel 2.3.

Es bietet sich daher an, Partnerschaften mit Dritt-Herstellern zu schließen, die eine Portfolioergänzung und die Verfolgung von Markttrends ermöglichen. Darüber hinaus wird nicht nur die Netzbelegung gesichert, sondern auch eine Risikoteilung gewährleistet.

Vor diesem Hintergrund möchte das 3rd Party Enabling Programm der T-Mobile den Value Added Service Providern zur Entwicklung und Vermarktung von mobilen Diensten und M-Commerce Produkten anregen und befähigen. Auf Basis von Mobilfunkstandards (z.B. WAP) und T-Mobile Vorprodukten, sogenannter Enabling Produkte, sollen Applikationen und Dienste für Nutzer des T-Mobile Netzes in Deutschland entwickelt werden.

2.3 Enabling Produkte

Im Zentrum der entwickelten Simulations-Anwendung stehen die Enabling Produkte (Vorprodukte). Dabei handelt es sich um Dienste, die kunden- oder netzspezifische Informationen (z.B. Positionsdaten eines Kunden) zur Verfügung stellen. Besonders beleuchtet werden in diesem Zusammenhang die standortbezogenen Dienste, deren Chancen und wirtschaftliche Bedeutung ausführlich diskutiert werden. Weniger Aufmerksamkeit wird hingegen der „Customer Identity Application“ gewidmet, die dem Value Added Service Provider lediglich Metainformationen liefert.

❖ Location Based Services (LBS)

„Location Based Services ist ein Sammelbegriff für ortsbezogene Dienstleistungen, die über die Lokalisierungsfunktion des mobilen Endgerätes möglich werden.“[5] Den Ortsinformationen des Teilnehmers wird eine besondere Bedeutung zugesprochen, und die darauf basierenden Anwendungen werden als eine der aussichtsreichsten Anwendungen mobiler Multimediadienste angesehen. Bis zum Jahr 2006 wird eine Steigerung auf 1,5 Milliarden Nutzer von

[5] IZT, Entwicklung und zukünftige Bedeutung mobiler Multimediadienste, S. 70

ortabhängigen Diensten weltweit vorhergesagt, was einer jährlichen Wachstumsrate von ca. 180% bezogen auf 72 Millionen Anwender im Jahr 2001 entspricht. Abbildung 2 zeigt eine rasante Umsatzentwicklung der standortbezogenen Dienste für den europäischen und nordamerikanischen Markt.

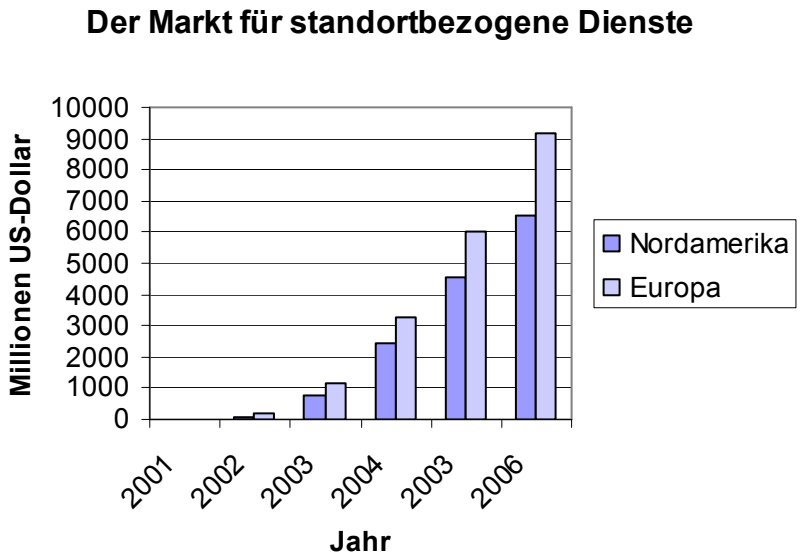

Abbildung 2 - Umsatzprognose für standortbezogene Dienste[6]

Mit Location Based Services wird das Handy zum Sextant des 21 Jahrhunderts. Die ortsbezogenen Dienste erlauben es Anwendungen zu entwickeln, die den aktuellen Aufenthaltsort eines Mobilkunden berücksichtigen und ihm so maßgeschneiderte Informationen und Dienstleistungen bereitstellen. Einige typische Anwendungsbeispiele sind:

- ***Next-to-Services***: Das mobile Internet bietet brandaktuelle Tipps zu nahegelegenen Restaurants und hilft bei der Online-Tischreservierung.

- ***Regionales Infotainment***: Per mobilem Internetdienst können die aktuellen Filmangebote des jeweiligen Aufenthaltsortes abgerufen, bestimmte Vorschauclips ausgewählt, nach freien Sitzplätzen gesucht und

[6] Seiler, Bernd (2002) Taschenbuch der Telekompraxis, Seite 15

schließlich die gewünschten Tickets reserviert und bezahlt werden.

- ***Tracking Services***: Über das mobile Internet hält das Handy seine Besitzer ständig auf dem Laufenden darüber, ob ein Freund zufällig in der Nähe ist. Damit sind spontane Verabredungen auf ein Bier oder fürs Kino genauso denkbar wie Handy-vermittelte „Blind Dates".

Die Genauigkeit bei der Standortangabe ist abhängig von der eingesetzten Technik. Gegenwärtig wird die Position am häufigsten auf der Basis der Funkzelle, über die eine Verbindung aufgebaut wurde, ermittelt. In diesem Fall ist für die Präzision der Ortung die Größe der jeweiligen Zelle maßgebend. Die Radien der Funkzellen reichen dabei von wenigen hundert Metern in Ballungszentren bis zu mehreren Kilometern in ländlichen Gebieten.

Bei T-Mobile können die ortsbezogenen Dienste mittels verschiedener Trägerdienste genutzt werden. Neben GPRS und CSD ist auch eine Nutzung per SMS möglich. Dazu sendet der Benutzer eine SMS mit einem Suchbegriff an eine bestimmte Service-Nummer. In einer Antwort-SMS erhält der Kunde dann die gewünschten standortbezogenen Informationen. Eine andere Möglichkeit, welche jedoch in der Simulation nicht nachgestellt wird, ist „Voice-LBS". Hier werden die Standortdaten auf Basis der Sprache bereitgestellt. In naher Zukunft wird als weiterer Trägerdienst UMTS hinzukommen.

❖ **NetworkInfo Request**

Ein weiterer Dienst der Enabling Produkte ist der NetworkInfo Request. Im Gegensatz zu den Location Based Services, handelt es sich dabei nicht um ein umsatzträchtiges Geschäftsfeld, dessen wirtschaftliche Bedeutung diskutiert werden muss. Der Dienst stellt gegenüber Dritten lediglich Metainformationen bereit und hat vorwiegend einen informativen Charakter.

Um den Nutzen des Services verstehen zu können, muss zunächst der Aufbau eines Mobilfunknetzes betrachtet werden. Das Versorgungsgebiet eines

Mobilfunknetz ist geographisch in Zellen eingeteilt, wobei jede Zelle von einer eigenen Basisstation mit Funksignalen versorgt wird. Stark vereinfacht werden die einzelnen Zellen üblicherweise in Form von Sechsecken dargestellt.

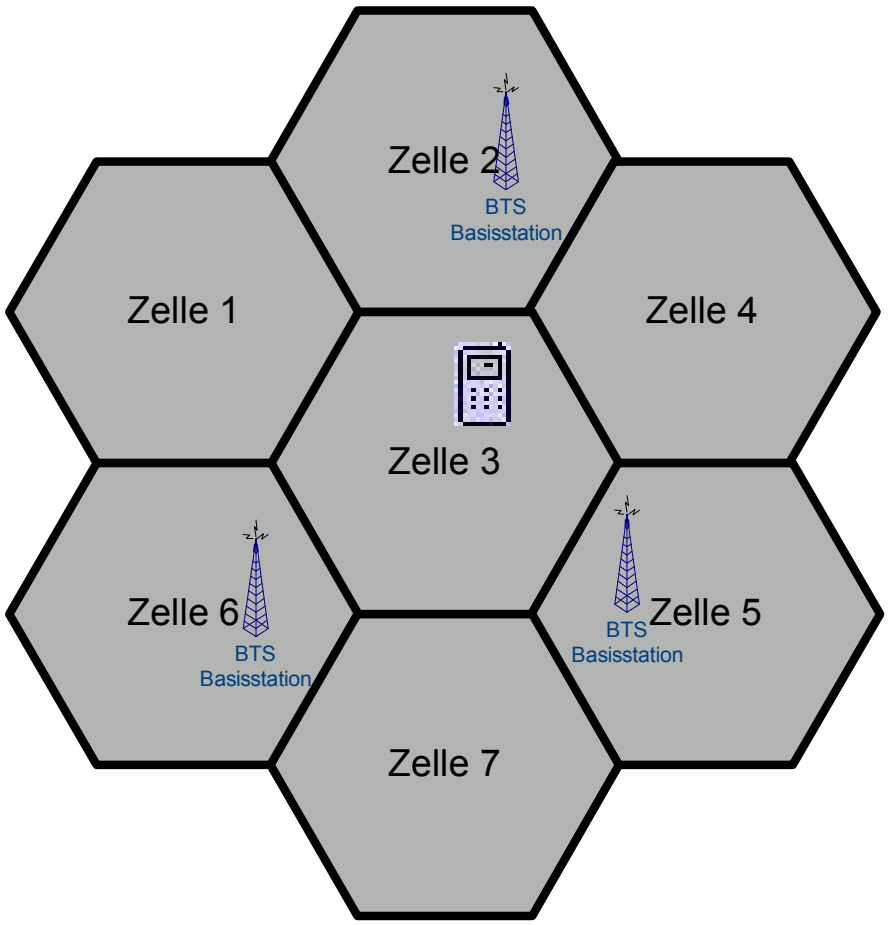

Abbildung 3 - Zellstruktur eines Mobilfunknetzes

Bei der Lokalisierung einer Mobilstation werden nicht direkt die geographischen Koordinaten ermittelt, sondern die Zelle, in der sich die Mobilstation gegenwärtig befindet. Die Aufgabe des NetworkInfo Requests besteht darin, zu einer übergebenen Zell-ID die zugehörigen geographischen Koordinaten zu ermitteln. Dies ist häufig für ausländische Kunden erforderlich, die sich durch das deutsche Mobilfunknetz(D1) bewegen. Die ausländische LBS-Plattform erhält in diesem Fall eine D1 Zell-ID, welche sie über den Network-Info Request in geograpische Koordinaten auflöst.

❖ **Identity Request**

Der Identity Request kann aus einem ähnlichen Blickwinkel wie der Location Request betrachtet werden. Auch hier stellt sich die Frage nach der wirtschaftlichen Relevanz der Informationen und wie sie von Dritten genutzt werden kann.

Aufgabe des Dienstes ist es, zu einer IP-Adresse die zugehörige MSISDN zu ermitteln. Ein VASP kann auf diese Weise einen Kunden identifizieren und ihm personalisierte Informationen zur Verfügung stellen. In Verbindung mit den Location Based Services wird es ferner möglich sein, dem Mobilfunk-Kunden ganz gezielt dienstliche und private Informationen zu senden. Ein mögliches Szenario wäre ein Benutzer, der ortsabhängige Informationen über einen VASP-Dienst abruft. Neben den schon bekannten Positionsdaten des Kunden, läßt sich nun über den Identity Request auch dessen Identität ermitteln. Liegen dem VASP für den identifizierten User bereits Profildaten vor, die Auskunft über dessen Vorlieben und Bedürfnisse geben, könnten ihm (dem User) entsprechende Angebote, z.B. über Elektroartikel oder Reisen, zugesendet werden. Durch die Verfolgung einer solchen „One-To-One“-Marketing-Strategie bieten sich dem stationären Einzelhandel vielversprechende Chancen gezielt Produktinformationen an den Endkunden zu senden. Vorraussetzung für die Identifizierung der Mobilstation ist allerdings, dass das Einverständnis des jeweiligen Kunden vorliegt.

Kapitel 3

Situationsbeschreibung

Wie bereits in der Einleitung erwähnt ist das Ziel der vorliegenden Arbeit, jenen Bereich eines Mobilfunknetzes zu simulieren, in dem sich die Enabling Produkte befinden. Das Kapitel widmet sich daher einer kurzen Beleuchtung der Architektur eines Mobilfunknetzes und beschreibt wo die zu simulierenden Enabling Produkte angesiedelt sind. Schließlich werden die Datendienste, mittels derer auf die Informationen zugegriffen wird, betrachtet.

3.1 Architektur eines Mobilfunknetzes

Grundlage der zu entwickelnden Simulationsumgebung ist das GSM (Global System for Mobile Communications) Mobilfunknetz. Es handelt sich dabei um einen europäischen Standard, der seit den frühen neunziger Jahren zur Verfügung steht. GSM wurde in erster Linie zur Sprachvermittlung entwickelt und löste als System der zweiten Generation die nationalen analogen Netze ab.

In diesem Abschnitt wird kurz auf die Netzstruktur und die Komponenten des GSM-Netzes sowie des internen IP-Netzes eingegangen, die für die Simulation von Bedeutung sind. Abbildung 4 zeigt das GSM-Netz sowie das dazugehörige interne IP-Netz (Service-Netzwerk).

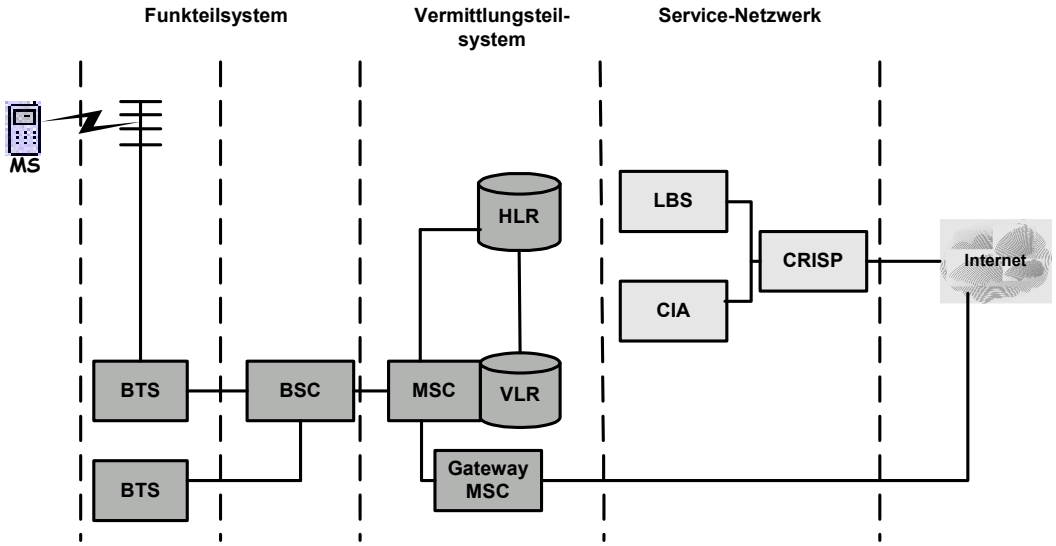

Abbildung 4 - Aufbau eines GSM-Netzes

3.1.1 Komponenten des GSM-Netzes

Mobilstationen (MS)

Unter einer Mobilstation versteht man das Datenendgerät, mit dem ein Mobilfunkkunde innerhalb eines Mobilfunknetzes kommuniziert. Typischerweise handelt es sich dabei um ein Handy.

Jede Mobilstation verfügt über eine SIM-Karte (Subscriber Identity Module) , auf der alle nutzerspezifischen Daten abgelegt sind.

Feststationensystem (BSS)

Feststationssysteme werden auch als Standortbereiche bezeichnet und umfassen alle Funktionen, die für eine dauerhafte Funkverbindung zu einer Mobilstation erforderlich sind. Darüber hinaus sind sie für die (De-)Codierung der Sprachdaten sowie für die Anpassung der Datenraten, die vom und zum drahtlosen Telephonienetz vorgenommen werden, verantwortlich.

Jede BSS besteht aus zwei Teilen:

- Sende/Empfangsstation (BTS)
  "Das BTS enthält die eigentliche Funktechnik, also Sender und Empfänger, auf der Netzseite. Basisstationen müssen in relativer großer Zahl an funktechnisch geeigneten Orten aufgebaut werden."[7]

  Eine BTS besteht aus verschiedenen funktechnischen Einrichtungen (Antenne, etc.) und kann eine Funkzelle bilden.

- Feststationensteuerung (BSC)
  "Die Basis- oder Feststationssteuerung führt die Vermittlung und steuert den Ablauf der Transmissions-Prozesse der Sende/Empfangsstationen (BTS´s). Dabei soll ein BSC mehrere BTS´s bedienen."[8]

Dienstvermittlungsstellen (MSC)

"Die MSC hat die Aufgabe, die Gesprächsverbindung zum gewünschten Teilnehmer aufzubauen und den Übergang zum Festnetz, aber auch zu anderen Mobilfunkvermittlungen zu ermöglichen". [9]

Gateway MSC (GMSC)

Das GMSC ist ein MSC, welches zusätzlich noch die Funktionalitäten übernimmt, die notwendig sind, um mit einem fremden Telefonienetz, z.B. einem anderen ISDN-Netz oder dem Internet, zu kommunizieren. Das GMSC fungiert also als Eingangs- und Ausgangstor zwischen dem eigenen GSM-Netz und allen anderen Telefonienetzen.

Heimatregister (HLR)

„Das HLR ist bezüglich der Teilnehmerdaten die zentrale Master-Datenbank. Die Heimatdatei übernimmt im Netz die administrativen Funktionen und beinhaltet die semipermanenten und temporären Daten aller Funkteilnehmer, die auf Dauer

[7] Janik, Blank, Telekommunikation für Profis, 10. Auflage, Seite 395
[8] Biala, Mobilfunk und Intelligente Netze, Seite 66
[9] Janik, Blank, Telekommunikation für Profis, 10. Auflage, Seite 396

dem HLA-Bereich (Home Location Area) zugeordnet sind“[10]. Diese Teilnehmerdaten dienen vor allem dem Verbindungsaufbau (Call-Setup) und der Service-Führung.

Besucherregister (VLR)

„Das Visitor Location Register kann als lokale Datenbank bezeichnet werden, mit einem geringen Anteil an Steuerungsfunktionen. Sie beinhaltet - bezüglich eines Teilnehmers - die Teilmenge seiner Daten, einschließlich des aktuellen Aufenthaltsortes, die für den Verbindungsaufbau von Bedeutung sind.“[11]

3.1.2 Komponenten des internen IP-Netzes

Customer Related Information for Service Provisioning (CRISP)

Der CRISP-Server bildet eine zentrale Schnittstelle für alle VASP-Anfragen nach der Lokalität bzw. der Identität eines Kunden. Als Bindeglied zwischen dem Mobilfunknetz einerseits und der zu bedienenden Anwendung andererseits, führt er zunächst die Authentifizierung und Autorisation des VASP durch und delegiert anschließend die Anfragen an die entsprechenden Enabling Produkte. Durch die Bündelung dieser Funktionen in einem eigenen Netzelement, werden die Dienste der Enabling Produkte auf Ihre Kernaufgaben reduziert und eine spätere Integration weiterer Dienste vereinfacht. Der CRISP ist ein logischer Server, der physikalisch aus einem CRISP-Webserver und einem CRISP-Applikationserver besteht.

Location Based Services (LBS)

Der LBS-Server stellt eines der beschriebenen Enabling Produkt dar. Er implementiert die Dienste „Location Based Services“ und „NetworkInfo Request“. Der LBS-Server nimmt die entsprechenden Anfragen vom CRISP-Server entgegen und verarbeitet diese. Anschließend übermittelt er die

[10] Janik, Blank, Telekommunikation für Profis, 10. Auflage, Seite 69

[11] Janik, Blank, Telekommunikation für Profis, 10. Auflage, Seite 70

gewünschten Daten oder eine Fehlermeldung zurück an den CRISP, der diese unverändert an den VASP weitergibt.

Customer Identity Application (CIA)

Das zweite Enabling Produkt ist der CIA-Server, der den gleichnamigen Dienst bereitstellt. Die Verarbeitungslogik erfolgt analog zu der des LBS-Servers.

3.2 Datendienste (Trägerdienste)

Der Siegeszug von Daten- und Multimedia-Diensten im Mobilfunkbereich war nur durch eine Ausweitung der verfügbaren Bandbreiten möglich. Diese wurde durch Erweiterungen des GSM-Netzes um Technologien wie HSCSD und GPRS geschaffen. In naher Zukunft wird mit der Einführung von UMTS, ein grundlegend neues System erwartet, das noch leistungsfähiger ist. Abbildung 5 zeigt die Entwicklung der verschiedenen Technologien in unterschiedlichen Regionen.

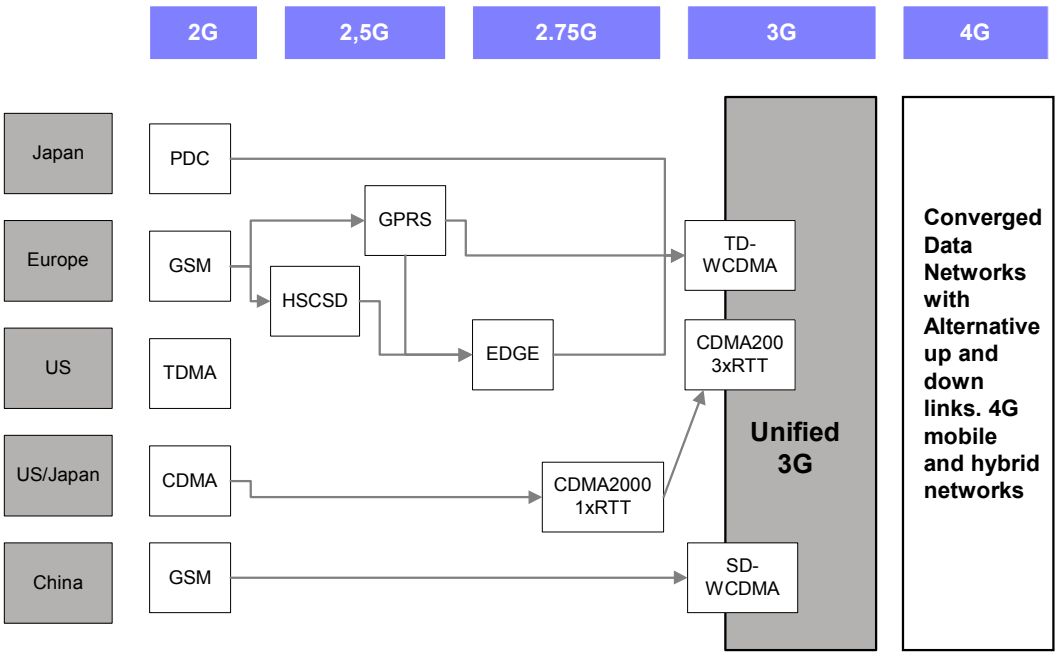

Abbildung 5 - Regionale Entwicklung der Trägerdienste

Zum Verständnis der vorliegenden Arbeit sind lediglich die im Folgenden beschriebenen Technologien von Bedeutung:

- **General Packet Radio Service (GPRS)**

  GPRS ist eine Übertragungstechnologie, die es ermöglicht Daten schnell über ein Mobilfunknetz zu übertragen. Die Technologie nutzt die Vorteile der paketorientierten Datenübertragung und der Kanalbündelung. Hierdurch ergibt sich theoretisch bei einem Datendurchsatz von 21,4 kBit/s pro Kanal und bei einer Bündelung der acht GSM-Kanäle eine maximale Datenübertragungs-Kapazität von 171,2 kBit/s. Dabei handelt es sich vorerst jedoch nur um einen theoretisch erreichbaren Wert. Praktisch liegt die Übertragungsrate lediglich zwischen 40 und 50 kBit/s.

  Ein weiterer Vorteil von GPRS ist, das die Tariffierung auf Basis des übermittelten Datenvolumens erfolgt. Gezahlt wird deshalb nur für die Menge der heruntergeladenen Daten und nicht für die in Anspruch genommene Zeit.

  Da die Daten in Paketen immer dann übertragen werden, wenn die Netzkapazitäten frei sind, kann es bei einer Überlastung des Netzes jedoch zu enormen Verzögerungen kommen.

- **Highspeed Circuit Switched Data (HSCSD)**

  Eine weitere neue Technologie, welche die Datenübertragung beschleunigt, ist HSCSD. Gegenüber einem herkömmlichen GSM-Kanal, der ein Übertragungsrate von 9,6 kbit/s erlaubt, erhöht ein HSCSD-Kanal den Datendurchsatz auf 14,4 kbit/s. Dies ist möglich durch ein Kodierungsprinzip, welches die GSM-Übertragungsrate um 50% je Kanal steigert. Die Leistungsfähigkeit von HSCSD liegt somit bei der Nutzung von vier Zeitschlitzen zwischen 38,4 und 57,6 kBits/s.

Der Vorteil von HSCSD für den Nutzer ist die feste Übertragungsbandbreite, welche während der Verbindung gleichmäßig zur Verfügung steht.

Kapitel 4

Anwendungsdefinition

Dieses Kapitel beschreibt zuerst die Anwendungsidee und aus welcher Notwendigkeit heraus diese entstanden ist. Daran anschließend schildert die Anforderungsspezifikation welche Dienste das System erbringen soll und unter welchen Randbedingungen es operieren muss. Zum Schluss werden die Anwendungsfälle, die sich aus der Sicht der Akteure ergeben, identifiziert und erläutert.

4.1 Anwendungsidee

Im Rahmen des T-Mobile Developer Centers wird die Entwicklung von Applikationen durch Dritt-Hersteller (3rd Party) unterstützt. Den 3rd-Party Entwicklern soll durch eine Reihe von Beispiel-Anwendungen ein erster Einblick in den Umgang mit den Enabling Produkten ermöglicht werden. Um die Entwicklung möglichst einfach zu gestalten, werden diese Anwendungen als auch Quelltext zur Verfügung gestellt.

Für die Nutzung dieser Beispiel-Applikation ist es erforderlich die jeweiligen Enabling Produkt APIs[12] bereitzustellen. Aus Gründen der Betriebssicherheit kann jedoch eine direkte Anbindung an das produktive Wirknetz der T-Mobile nicht erfolgen. Für einen Experimentierbetrieb soll stattdessen eine Art „Spielwiese“ für die Entwickler in Form eines Netzsimulators geschaffen werden. Ziel dieser Simulation ist es in erster Linie die Schnittstellen der Enabling Produkte exakt nachzubilden (Wirknetzsimulation). Eine Kopie der internen Architektur des realen Wirknetzes ist nicht vorgesehen.

[12] Application Programing Interface = Programmier- und Anwendungsschnittstelle

Der zu entwickelnde Netzsimulator sollte neben der Wirknetzsimulation auch eine Steuerungsoberfläche zur Simulation der Mobilstationen sowie eine VASP-Beispiel-Applikation beinhalten. Das folgende Kontextdiagramm zeigt die Elemente der Netzsimulators.

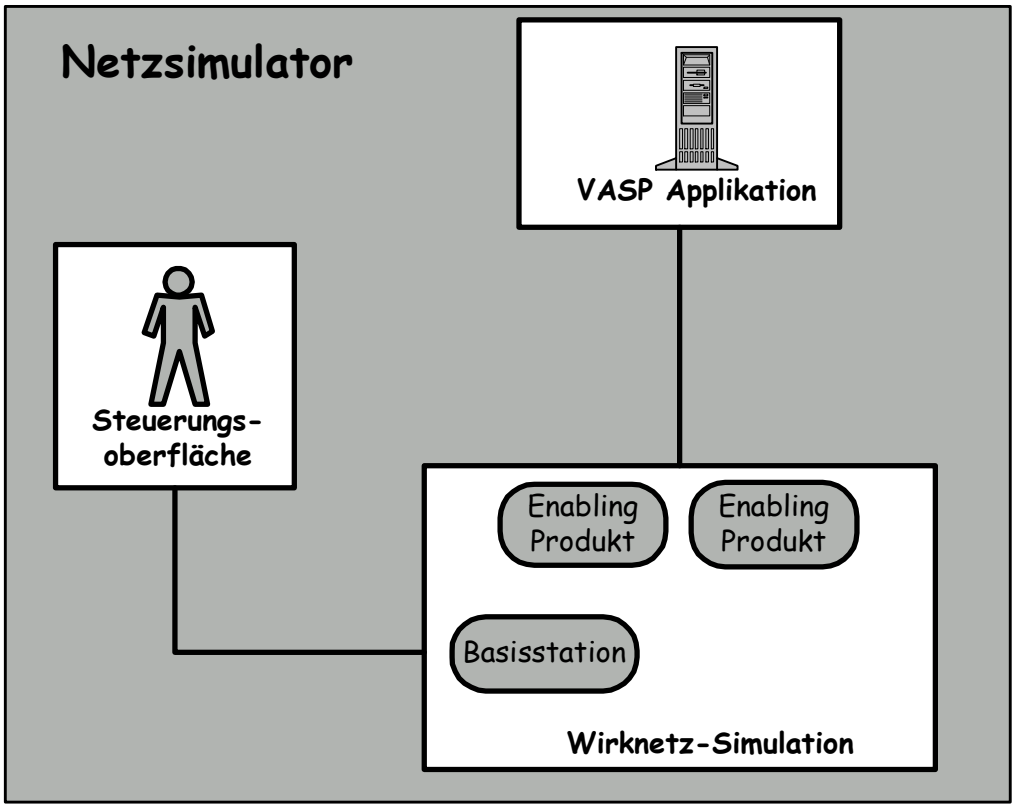

Abbildung 6 - Komponenten des Netzsimulators

4.2 Anforderungsspezifikation

4.2.1 Funktionale Anforderungen

An dieser Stelle wird beschrieben welche Funktionalität das System in Form von Diensten erbringen soll.

- **Wirknetz-Simulation**
  Kernstück der zu entwickelnden Anwendung ist die Simulation des Wirknetzes. Letzteres umfasst zunächst nur die Enabling Produkte „Location Based Services“ und „Customer Identity Applikation“. Das Hauptaugenmerk liegt dabei auf der Implementierung der Schnittstellen der jeweiligen Enabling Produkte. Das Resultat, das der Netzsimulator auf die Anfrage eines Value Added Service Providers liefert, muss mit der entsprechenden Antwort des Live-Systems übereinstimmen. Die internen Abläufe des realen Wirknetzes sollen nicht nachgestellt werden.

- **Steuerungstool**
  Zur Darstellung der Mobilfunkkunden bzw. der Mobilstationen muss neben der Wirknetz-Simulation ein weiteres Element, das Steuerungstool, implementiert werden. Primäre Aufgabe des Tools ist die Verwaltung der Mobilstationen sowie die Abbildung der Funktionen „GPRS-Call“, „CSD-Call“ und „SMS-Versand“. Das Steuerungstool sollte eine grafische Oberfläche haben und als eigenständige Applikation laufen.

- **VASP-Applikation**
  Zur Vervollständigung des Gesamtbildes wird außerdem eine VASP-Anwendung implementiert. Die Applikation ermöglicht es die verschiedenen Dienste der Wirknetz-Simulation zu testen. Darüber hinaus dient sie dem VASP als Leitfaden für eine eigene Anwendungsentwicklung.

- **Sicherheitsaspekt**
  Zwischen dem Wirknetz und dem VASP werden Lokalisierungsinformationen von Mobil-Kunden ausgetauscht. Bei dieser Art von Information handelt es sich um kundenspezifische Daten, die an Dritte aus datenschutzrechtlichen Gründen nicht weitergereicht werden dürfen. Aus diesem Grund muss die Kommunikation zwischen dem Wirknetz und dem VASP über eine gesicherte Verbindung erfolgen.

  An eine sichere Verbindung werden hinsichtlich der Daten folgende

Anforderungen gestellt:

1. *Vertraulichkeit der Daten*: Aus abgehörten Daten kann ein Angreifer den eigentlichen Inhalt nicht ermitteln.

2. *Integrität der Daten*: Eine Verfälschung der Daten kann nicht unerkannt erfolgen.

3. *Authentizität der Daten*: Die übertragenen Daten stammen tatsächlich vom Absender, d.h. zusätzliche unechte Pakete werden erkannt.

- **Fehlertracking**
  In der Vergangenheit kam es zwischen VASP und Wirknetz oft zu Integrationsherausforderungen beim Verbindungsaufbau. Aber auch die Nutzung der einzelnen Dienste bzw. Enabling Produkte bereite durch falsche Parametrisierung Schwierigkeiten. Die Wirknetz-Simulation sollte daher über ein Fehlertracking verfügen, welches es dem VASP ermöglicht eigenständig Fehlerquellen aufzuspüren.

- **Multi-User Fähigkeit**
  Der Netzsimulator sollte Multi-User fähig sein. Hierdurch kann beim VASP eine Serverinstallation vorgenommen werden, die von mehreren Entwicklern gemeinsam genutzt wird. Denkbar ist außerdem, das ein T-Mobile Experte sich auf eine VASP-Installation einloggt und Prozesse mitverfolgt, wodurch die Support-Möglichkeiten verbessert werden.

4.2.2 Nichtfunktionale Anforderungen

Die im Folgenden aufgeführten Schlagworte beschreiben unter welchen Randbedingungen der Netzsimulator bzw. dessen Entwicklungsprozess operieren muss. Sie definieren die Qualität der im vorigen Kapitel beschriebenen Dienste.

- Plattformunabhängigkeit
  Mit wachsender Anzahl der Value Added Service Provider steigt auch die Heterogenität der Systeme, auf denen 3rd-Party Entwickler ihre Applikationen erstellen. Vor diesem Hintergrund ist es von entscheidender Bedeutung eine plattformneutrale Simulationsanwendung zu entwickeln.

- Komponentenbasiert
  Für eine kostengünstige und zeitnahe Weiterentwicklung müssen Teile der Software wiederverwendet werden können. Das Gesamtsystem darf daher nicht als ein monolithischer Block konzipiert und entwickelt werden, sondern sollte sich in einzelne Teile zerlegen lassen. Die dabei entstehenden Komponenten lassen sich dann gegebenenfalls auch von Drittherstellern für eigene Anwendungen wiederverwenden. Weiterhin können einzelne Komponenten bei Bedarf durch verbesserte ausgetauscht werden.

- Aktualisierbarkeit/Wartbarkeit der Schnittstellen
  Das Wirknetz der T-Mobile ist einem ständigen Entwicklungsprozess unterworfen. Innerhalb eines Jahres sind mehrere Versionen (Releases) eines Enabling Produktes möglich. Um den Drittherstellern einen reibungslosen Weg in die Produktivumgebung zu ebnen, ist es daher wichtig den Netzsimulator aktualisieren zu können. Die Schnittstellen des Netzsimulators sollten modifiziert werden können, ohne das eine Anpassung des Quellcodes erforderlich ist.

- Erweiterbarkeit des Systems
  Da es kaum möglich ist, die umsatzträchtigen mobilen Killer-Applikationen der Zukunft vorauszusagen, ist es für die Netzbetreiter von entscheidender Bedeutung, sich darauf einzurichten, schnell und kostengünstig neue Dienste anbieten und so auf Marktrends reagieren zu können. Neben den bestehenden Enabling Produkten ist in Zukunft daher mit weiteren Diensten zu rechnen, die von der T-Mobile den Drittherstellern angeboten werden. Folglich muss gewährleistet werden, das sich neue Enabling Produkte einfach in das

entwickelte System integrieren lassen.

- Mehrschichtige Architektur
  Im Hinblick auf die mögliche Weiterentwicklung des Netzsimulators zu einem realitätsnahem Testsystem, ist es sinnvoll die Anwendung als Multi-Tier Architektur zu konzipieren. Hierdurch können die verschiedenen Schichten gemäß der Produktivumgebung auf unterschiedliche Systeme verteilt werden.

4.3 Anwendungsfälle

In diesem Kapitel werden verschiedene Anwendungsfälle aufgeführt, die das externe Systemverhalten aus der Sicht eines VASP und eines Mobilfunk-Kunden ausdrücken. Sie beschreiben was das System leisten soll und zeigen die benötigten Interaktionen zwischen dem System und den Akteuren auf. Das folgende *Use Case* Diagramm zeigt die identifizierten Anwendungsfälle zusammen mit den beteiligten Akteuren.

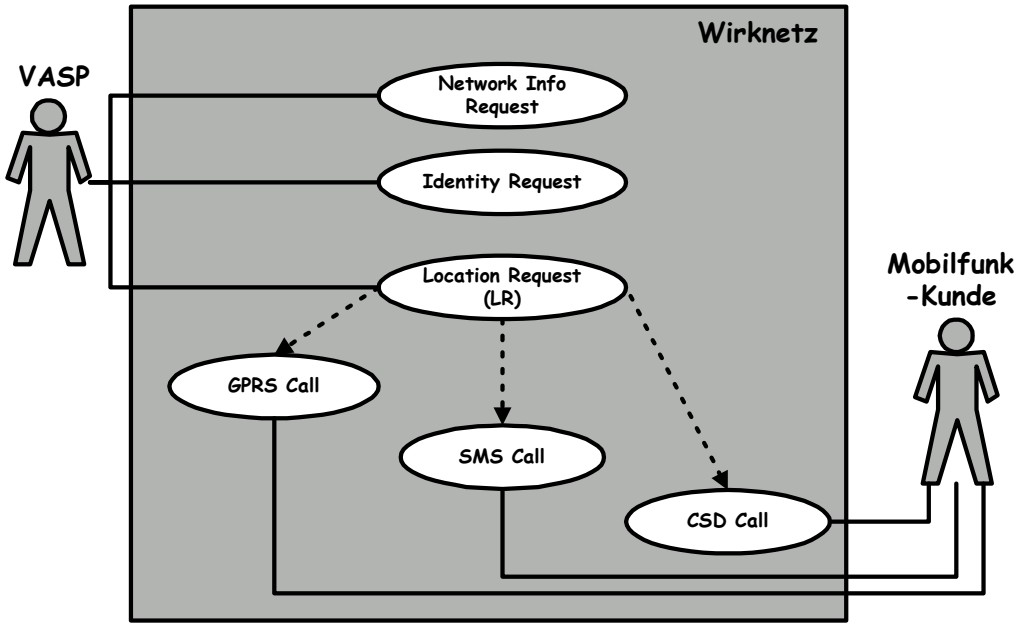

Abbildung 7 - UseCases des Wirknetzes

Wie bereits in Kapitel 2.3 erläutert kann ein Mobilfunkteilnehmer sich über verschiedene Datendienste in das Mobilfunknetz einwählen. Infolgedessen ergeben sich aus der Sicht des Mobilfunkkunden die folgenden Anwendungsfälle:

- **CSD-Call**

  Wählt eine Mobilstation sich über den Datendienst „CSD“ in das T-Mobile Datennetz ein, so werden durch das IN[13]-System Einwahlinformationen, wie Zell-ID, CallingParty[14] und CalledParty[15] im LBS-Server zwischengespeichert um für spätere VASP-Anfragen zur Verfügung zu stehen. Danach erfolgt die Zuweisung einer IP-Adresse an die Mobilstation. Anschließend kann der Mobilfunkkunde durch die Angabe einer URL, Informationen aus dem Internet bzw. von einem VASP herunterladen.

- **GPRS-Call**

  Beim Verbindungsaufbau mit GPRS wird ein PDP[16]-Context aufgebaut. Dieser enthält den verwendeten Adresstyp, die PDP (meist IP) Adresse der Mobilstation und die verlangte Qualität. In paketorientierten Systemen (PS-Systemen) existiert keine IN-Integration, wodurch im LBS-Server auch keine Einwahldaten hinterlegt werden. Diese Informationen werden bei jedem Location-Request vom LBS-Server im VLR abgefragt und gegebenfalls im VLR durch den Versand einer SMS an die Mobilstation aktualisiert. Ist der Verbindungsaufbau gelungen, kann über eine URL auf Ressourcen aus dem Internet zugegriffen werden.

- **SMS-Call**

  Das Versenden einer SMS ähnelt dem GPRS-Verfahren. Durch den Versand der SMS sind die Positionsinformationen der Mobilstation jedoch

[13] IN = Intelligent Network
[14] CallingParty = Rufnummer des Anrufenden
[15] CalledParty = Rufnummer des Angerufenen
[16] PDP = Packet Data Protocol

stets aktuell. Die MSISDN[17] an welche die SMS verschickt wurde, ist mit einem bestimmten Dienst verbunden (z.B. Wetterinformationen). Ein VASP erhält die SMS, ermittelt die gewünschten Informationen und sendet eine SMS als Antwort zurück.

Der VASP nutzt die Dienste der jeweiligen Enabling Produkte, um kunden- oder netzspezifische Informationen zu erhalten. Die dafür relevanten Anwendungsfälle sind in Abbildung 7 dargestellt und werden im Folgenden beschrieben:

- **Location Request**

  Der Location Request ermittelt die Lokalität einer Mobilstation und gibt diese Information im gewünschten Positionierungsformat (z.B. Point oder Rectangle) an den VASP weiter. Hinsichtlich des vom Mobilfunkkunden verwendeten Datendienstes, ergeben sich unterschiedliche Prozedurabläufe zur Bestimmung der Position. Diese Abläufe werden in Kapitel 6.3.4 näher beschrieben.

- **Network Info Request**

  Der Network Info Request ist ein Dienst der vorwiegend von anderen Operatoren (z.B. VIAG oder ausländischen Partnern) beansprucht wird. Ein mögliches Szenario wäre ein Mobilfunkkunde eines ausländischen Partnerunternehmens, der sich im T-Mobile Datennetz befindet und einen ortsabhängigen Dienst nutzen möchte. Das fremde Mobilfunkunternehmen erhält daraufhin ein Zell-ID (aus dem T-Mobile Datennetz), die ihr nicht bekannt ist. Um diese Information in geographische Koordinaten auflösen zu können, nutzt der Operator den Network Info Request.

- **Customer Identity Request**

  Erhält ein VASP eine Anfrage von einem Mobilfunkkunden, so ist ihm

[17] MSISDN = Mobilstation ISDN

meistens nur die IP-Adresse bekannt. Da es sich dabei gewöhnlich um eine dynamische Adresse handelt, kann diese nicht als langfristiges Identifikationsmerkmal verwendet werden. In diesem Fall kann der VASP über den Customer Identity Request für eine IP-Adresse die entsprechende MSISDN erfragen. Vorraussetzung hierfür ist jedoch, dass das Einverständnis des Mobilfunkkunden vorliegt.

Kapitel 5

Technologien

An dieser Stelle werden die dem System zugrunde liegenden Technologien beleuchtet. Im Mittelpunkt des Kapitels steht dabei die J2EE-Technologie, die bei der Umsetzung der in Kapitel 4.1 beschriebenen Anwendungsidee, eine zentrale Rolle spielt. Da J2EE-Systeme vorwiegend aus mehreren Schichten bestehen, werden zunächst die Vorzüge mehrschichtiger Systeme dargestellt. Nach der Betrachtung verschiedener Java-Technologien, wie z.B. JMS, JSP, erfolgt zum Schluss ein kurzer Einblick in das Thema „SSL“.

5.1 Mehrschichtige Systeme

Mehrschichtige Systeme (Multi-Tier) zeichnen sich vor allem durch die Unabhängigkeit und Austauschbarkeit der einzelnen Schichten aus. Jede Schicht ist ein in sich geschlossenes System, das über definierte Schnittstellen ansprechbar ist.

Das Schichtenprinzip bieten gegenüber ungeschichteten Modellen folgende Vorteile.

Skalierbarkeit

In einem Multi-Tier Model verbinden sich die Clients nicht direkt mit einem Datenbanksystem sondern mit einem Applikationsserver. Letzterer verfügt über einen Pool an Datenbank-Connections wodurch die Datenbank entlastet wird. Darüber hinaus kann durch den Einsatz mehrer Applikationsserver die Last weiter verringert werden (Load Balancing).

Multiple Clients

Ein mehrschichtiges Model führt zu schlankern Clients, da sich der größte Teil der Programmlogik in der Middle-Tier befindet. Dies führt zu einer höheren Anzahl einsetzbarer Client-Plattformen.

Netzwerk-Effizienz

Ein Multi-Tier Model reduziert die Netzwerklast. Statische Informationen können im Application-Server zwischengespeichert werden und müssen nicht für jeden Client-Zugriff erneut angefragt werden.

Losgelöste Geschäftslogik

Die Geschäftslogik kann in der Applikationsserver-Schicht verändert werden ohne das dies Einfluss auf den Client hat. Die Änderung wird an einer zentralen Stelle vorgenommen und muss nicht für jeden Client wiederholt werden.

Es gibt verschiedene Modelle mehrschichtiger Anwendungsarchitekturen. In der Praxis trifft man am häufigsten auf das Middle-Tier Modell. Dies besteht aus den drei Schichten Client-Tier, Middle-Tier und EIS-Tier. Das hier entwickelte System basiert auf einer Java 2 Enterprise Edition Architektur (J2EE). Bei dieser Architekturvariante erfolgt eine Trennung zwischen der Präsentationslogik und der Geschäftslogik, wodurch der Management Overhead reduziert und die Wiederverwendbarkeit erhöht werden. Die Middle-Tier wird also weiter in eine Presentations-Tier, eine Business-Tier und eine Integration-Tier unterteilt. Eine J2EE-Architektur besteht damit aus den folgenden fünf Schichten.

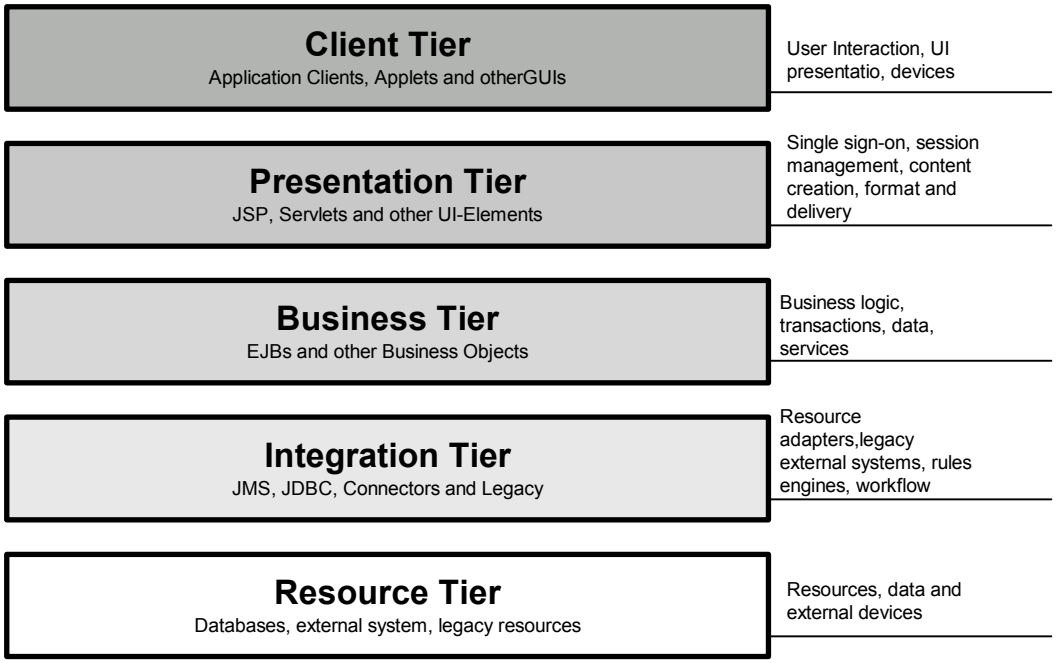

Abbildung 8 - Schichten einer J2EE-Architektur

Client-Tier

In dieser Schicht befinden sich alle Geräte oder Anwendungen, die auf das System zugreifen. Dabei kann es sich beispielsweise um einen Browser, ein Mobiltelefon oder einen PDA handeln.

Bei J2EE unterscheidet man zwischen webbasierten oder nicht webbasierten Applikationen. Bei einer webbasierten J2EE-Applikation läuft ein Browser in der Client-Tier, der statische oder dynamische HTML-Seiten von einem Webserver herunterlädt. Die dynamischen Seiten werden durch JSP-Pages oder Servlets, die innerhalb der Web-Tier des Webservers laufen, generiert.

Eine nicht webbasierte Applikation, wie z.B. ein Standalone-Client oder ein Applet, läuft in der Client-Tier und greift direkt auf die Enterprise-Beans in der Business-Tier zu, ohne über die Web-Tier zu gehen.

Web-Tier

In der Web-Tier wird die Präsentationslogik gekapselt. Durch diese Separierung kann man sehr flexibel unterschiedliche Ausgabeformate realisieren. So können beispielsweise neben HTML erzeugenden Komponenten, einfach weitere Module hinzugefügt werden, die z.B. WML- oder PDF-Dateien generieren, ohne das die Geschäftslogik hierfür verändert werden muss.

Bei den J2EE-Web-Komponenten handelt es sich um JSP-Pages, webbasierte Applets oder Servlets.

Business-Tier

In dieser Schicht befindet sich die Geschäftslogik. Sie bildet in der Regel den komplexesten Teil einer Anwendung. Durch die Auslagerung in eine eigene Schicht sowie der Trennung von der Präsentationslogik und den Daten, ergibt sich eine hohe Wiederverwendbarkeit.

Die Verarbeitung der Geschäftslogik erfolgt in J2EE durch Enterprise-Beans. Eine Enterprise-Bean erhält Daten von einem Client, verarbeitet diese und sendet sie zum Speichern an das Enterprise Information System (EIS). Umgekehrt kann eine Enterprise-Bean die Daten aber auch vom Enterprise Information System erhalten, verarbeiten und zum Client senden.

Integration-Tier

Ziel der Integration-Tier ist es eine einheitliche Schnittstelle zwischen der Business-Tier und der Resource-Tier herzustellen. Der Nutzen dieser Schicht wird deutlich wenn man bedenkt, dass Informationen nicht ausschließlich aus einer zentralen, neu angelegten Datenbank kommen können. Oft gilt es die Daten vorhandener System, wie z.B. ERP[18]-Anwendungen, in die eigene Applikation zu integrieren. Vor allem im B2B-Bereich ist es denkbar das Informationen von Geschäftspartner abgebildet werden sollen. Um dies zu erreichen, müssen die

[18] **E**nterprise **R**esource **P**laning (z.B. SAP)

Ressourcen der Business-Tier als API oder über standardisierte Protokolle zugänglich gemacht werden.

Mit der „J2EE Connector Architecture (JCA)“ wird man diesen Ansprüchen gerecht. Diese Architektur ermöglicht es eine J2EE-Plattform mit verschiedenen Enterprise Information Systems zu verbinden.

Resource-Tier

Die Resource-Tier ist die unterste Ebene im Schichtenmodell und beinhaltet die eigentliche Datenspeicherung. Sie kann beispielsweise eine Datenbank, ein ERP-System oder verschiedene Legacy-Systeme repräsentieren.

5.2 Java

5.1.1 Historie

Die von Sun Microsystems konzipierte Programmiersprache Java hat seit Mitte der neunziger Jahre eine rasante Entwicklung und Verbreitung erlebt. Sie hat sich im Ausbildungs- wie im kommerziellen Bereich als Standard etabliert. Die Anfänge von Java liegen hingegen in einem portablen Interpreter namens Oak, der Anfang der neunziger Jahre für die Entwicklung von Programmen für Kleingeräte (Radios, Videorecorder, etc) konzipiert wurde. Hinsichtlich der Zielplattformen spielten Eigenschaften wie Portabilität, Robustheit sowie eine geringe Programmgröße bereits eine entscheidende Rolle. Da die Nachfrage nach Java als Programmiersprache für kleine Elektrogeräte gering war und das World Wide Web zur gleichen Zeit einen Evolutionssprung erfuhr, änderte SUN Microsystems die Strategie für Java. Man erkannte die Bedeutung einer plattformunabhängigen Programmiersprache, mit der man Programme erstellen konnte, die sich auf einfache Weise über verschiedene Hardware- und Betriebssystemplattformen im Internet verteilen ließen. Die ursprünglichen Anforderungen kamen der neuen

Ausrichtung zugute. Hierzu gehört neben der Portierbarkeit beispielsweise die geringe Größe von Java-Programmen, die eine Verteilung über das World Wide Web in einer angemessenen Zeit möglich macht.

5.1.2 Eigenschaften von Java

Die wohl bestechendste Eigenschaft von Java ist die Plattformunabhängigkeit. Im Gegensatz zu vielen gängigen Programmiersprachen wie z.B. C oder C++, werden die Quelldateien vom Compiler nicht direkt in den Maschinencode übersetzt, sondern zunächst in einen Byte-Code gewandelt. Letzterer wird dann bei der Ausführung des Programms von der Java Virtual Maschine interpretiert und in den jeweiligen Maschinencode transformiert wird. Auf diese Weise lassen sich einmal erzeugte Anwendungen auf den unterschiedlichsten Betriebssystemen bzw. Geräten ausführen.

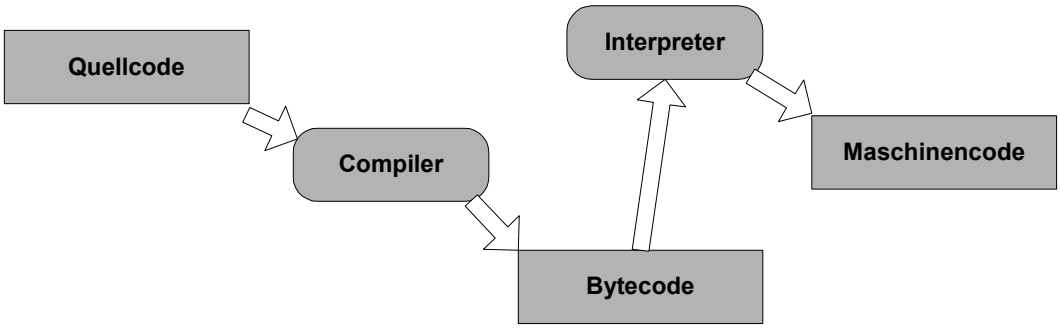

Abbildung 9 - Java Entwicklungsprozeß

Neben der Plattformunabhängigkeit kann Java durch die folgenden Schlagworte charakterisiert werden.

- **Einfach** - Java ist leicht für C++-Programmierer zu erlernen, da Java eine ähnliche Syntax wie C++ hat. Einige Eigenschaften, die in C++ zu kompliziert sind, wurden weggelassen, andere, die in C++ umständlich gelöst sind, hinzugefügt.
- **Objektorientiert** - Java unterstützt einfache Vererbung und Schnittstellen, jedoch keine Mehrfachvererbung. Es können abstrakte Klassen und

Methoden erklärt werden, außerdem final-Klassen, von denen keine weiteren Klassen ableitbar sind.

- **Verteilt** - Java-Anwendungen lassen sich durch Unterstützung der Technologien RMI und CORBA leicht auf mehrer System verteilen.
- **Interpretiert -** Java Programme liegen als Bytecode vor, der von der JVM interpretiert wird.
- **Robust -** Automatisches Speichermanagement und Speicherbereinigung (Garbage Collection) machen Java robust. Hinzu kommt eine strenge Typisierung der Sprache.
- **Thread-Unterstützung -** Ein paralleles Abarbeiten unterschiedlicher Aufgaben bringt Geschwindigkeitsvorteile. Threads können durch die Ableitung einer neuen Klasse einfach realisiert werden.
- **Dynamisch -** Werden in einer Superklasse neue Methoden oder Instanzvariabeln eingeführt, ist eine Anpassung der Subklassen nicht erforderlich. Außerdem wird der Speicherplatz automatisch freigegeben.
- **Sicher -** Der Bytecode schützt vor Verfälschungen und Virenbefall. Darüber hinaus ist ein Dateizugriff nur in einer begrenzten Weise möglich.

Ein gravierender Nachteil von Java ist die Tatsache, das interpretierte Sprachen eine geringe Ausführungsgeschwindigkeit haben als Compilersprachen. Die Ursache hierfür liegt in der Notwendigkeit begründet, das der Quellcode von der Virtual Maschine zunächst in den betriebssystemspezifischen Maschinencode übersetzt werden muss. Gegenüber anderen interpretierten Sprachen, wie z.B. Perl, hat Java jedoch den Vorteil, das die Syntaxüberprüfung nicht erst zur Laufzeit, sondern bereits beim Kompilieren erfolgt. Außerdem liegt der Bytecode bereits in komprimierter Form vor, was zusätzlich zu einer verbesserten Performance führt.

5.1.3 Servlets

Die Anforderungen an eine Internetanwendung sind in Zeiten des M-Commerce enorm hoch. Die Darstellung einfacher, statischer Webinhalte reicht längst nicht mehr aus. Eine moderne Webapplikation muss daher über einen hohen Grad an Dynamik und Interaktionsmöglichkeiten verfügen. Um solchen Ansprüchen gereicht zu werden, ist der Einsatz serverseitiger Programmiertechniken unabdingbar. Laut Rossbach gibt es hierfür drei verschiedene Ansätze[19]. Die wohl bekannteste Möglichkeit Webseiten Interaktivität zu verleihen ist das „Common Gateway Interface“ (CGI). Es definiert eine Schnittstelle zum Webserver und erweitert dessen Funktionalität. Die CGI-Technik ist jedoch veraltet und weist insbesondere hinsichtlich der Betriebssystemressourcen gravierende Nachteile auf. Ein weitere Ansatz sind zwei proprietäre Technologien. Dies ist zum einem Microsofts ISAPI, zum anderen das von Netscape entwickelte NSAPI. Beide Technologien konnten sich jedoch aufgrund der fehlenden Kompatibilität zu den gängigen Webserver-Produkten nicht durchsetzen. Eine echte Alternative und mittlerweile stark verbreitete Technologie stellen Servlets dar. Servlets sind im Speicher liegende Java-Programme, die in einem Servlet-Container (z.B. Tomcat) über eine JVM laufen. Sie zeichnen sich gegenüber der CGI-Technologie vor allem durch eine hohe Performance und Portierbarkeit aus. Die Gründe für die Portierbarkeit wurden bereits ausführlich im vorigen Kapitel behandelt. Der Performancevorsprung liegt darin begründet, das gegenüber den traditionellen CGI[10] Skripts nicht für jeden HTTP Request ein neuer Prozess gestartet. Stattdessen erzeugt die JVM[11] für jeden Request einen eigenen *Lightweight Java Thread* und keinen neuen Betriebssystemprozess. Abbildung 10 verdeutlicht diese Verfahrensweise anhand des Lebenszyklus eines Servlets.

[19] Vergleiche Rossbach, Java Server und Servlets, Seite 29

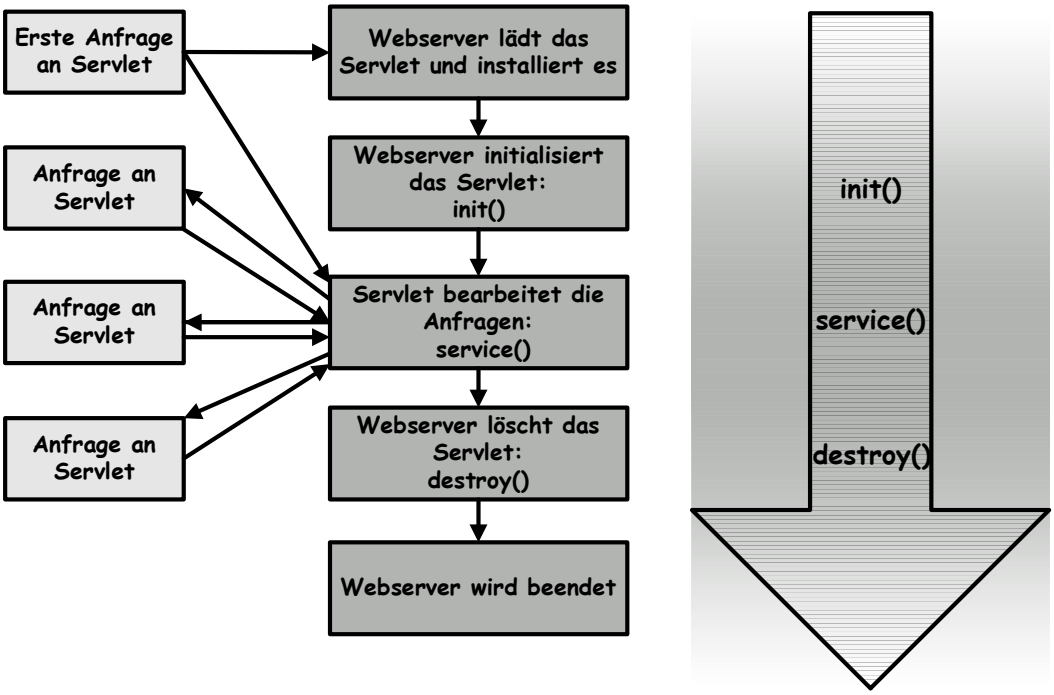

Abbildung 10 - Lebenszyklus eines Servlets

Ein Servlet besitzt einen genau definierten Lebenszyklus, dessen Ablauf vom Servlet Container gesteuert und überwacht wird. Bei einer Client-Anfrage wird zunächst überprüft ob eine Instanz des jeweiligen Servlets bereits im Container existiert. Ist dies nicht der Fall, so wird die Klasse geladen und instanziert. Anschließend wird die Initialisierungsmethode des Servlets aufgerufen. In dieser Methode werden Variablen initialisiert, die sich nicht mehr ändern und für alle weiteren Anfragen verwendet werden. Dabei kann es sich beispielsweise um eine Datenbankverbindung oder das Einlesen einer Property-Datei handeln. Konnte das Servlet ordnungsgemäß initialisiert werden, steht es nun bereit, einen HTTP Request[20] entgegen zu nehmen. Damit ein Servlet einen HTTP Request entgegen nehmen kann, wird vom Servlet Container die Methode *service()* aufgerufen. Dieser Methode wird das HTTP Request- Objekt und das HTTP Response-Objekt als Parameter übergeben. Nun kann die Abarbeitung des Servletcodes erfolgen und anschließend die vom Servlet generierte Antwort an den Client gesendet werden. Schließlich wird das Servlet für eine in der Serverkonfiguration

[20] Anfrage eines Clients

angegebene Dauer im Speicher gehalten und steht somit weiteren Anfragen zur Verfügung.

5.1.4 Java Server Pages (JSP)

Mit der Java Server Pages (JSP) Technologie ist eine weitere serverseitige Technologie entworfen worden, die eine schnelle und effiziente Entwicklung von dynamischen Webapplikationen möglich macht. Im Unterschied zu den Servlets wird hier der Javacode in die HTML-Seite eingebettet. Eine Java Server Page bietet daher die Möglichkeit, dynamischen Inhalt mit statischem Inhalt einer HTML Seite zu mischen. Vorteilhaft ist dies vor allem, wenn viele unterschiedliche Darstellungen einer Seite notwendig sind. Durch die von JSP bereitgestellten Kontrollstrukturen und Schleifen, kann ein Webdesigner eine Seite sehr mächtig und flexibel gestalten. Zu viel Javacode in einer Webseite birgt jedoch auch die Gefahr, dass die Seite unübersichtlich und schwer pflegbar wird. Idealerweise befindet sich daher die Anwendungslogik in JavaBeans, die direkt oder über JSP eigene Tags angesprochen werden. Das Tag-Konzept eignet sich hervorragend um eine Seite zu modularisieren. Die Tags können Attribute enthalten, so dass unterschiedliche Ausgaben hervorgerufen werden können. Ein Designer muss letztlich, neben seinen HTML-Kenntnissen, nur wissen wie er an die Daten der Beans kommt. Mit den JSP-Tags wird dem Webdesigner eine vertraute Technologie mit auf den Weg gegeben.

Java Server Pages benötigen eine JSP Engine die eine Umsetzung der JSP-Seite in ein Servlet durchführt und die Ergebnisseite als HTML zurückliefert.

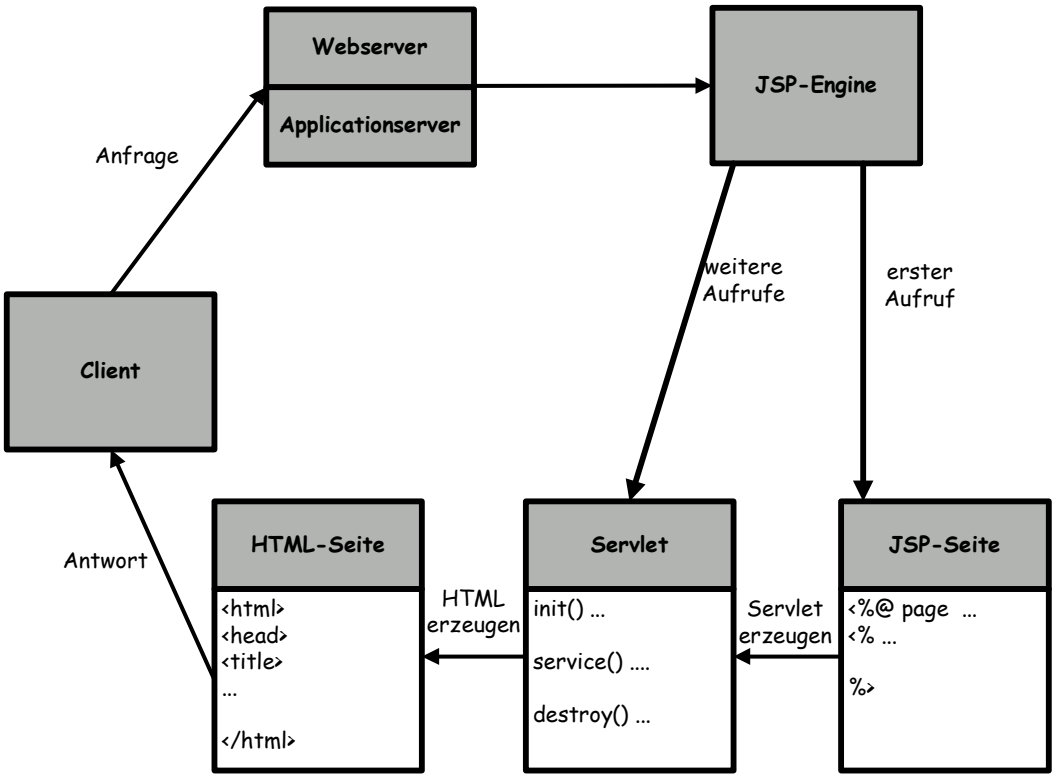

Abbildung 11 - Ablauf eines JSP-Requests[21]

Beim ersten Seitenaufruf generiert der Server automatisch ein Servlet im Hintergrund, das eine statische Seite mit dem gewünschten Seiteninhalt erzeugt. Das so erzeugte Servlet kann bei weiteren Aufrufen der Seite wiederverwendet werden, solange die JSP-Seite nicht verändert wird. JSP-Seiten können auch Quellcode in anderen Sprachen enthalten, diese werden über den Parameter „language“ eingestellt. Es muss allerdings eine Sprache gewählt werden die vom verwendeten JSP-Compiler unterstützt wird.

Gegenüber anderen vergleichbaren Technologien, wie z.B. ASP oder PHP, kann sich JSP vor allem durch eine bessere Performance behaupten. Der wesentliche Unterschied besteht darin, dass JSP zur Laufzeit nicht den Quellcode, sondern den bereits kompilierten Bytecode interpretiert.

[21] Volker Turau, Mark Schmidt, Java Server Pages und J2EE, Seite 51

5.1.5 Java 2 Enterprise Edition (J2EE)

5.1.5.1 Die J2EE-Plattform

Die Java 2 Enterprise Edition (J2EE), definiert einen Standard zur Entwicklung von Multi-Tier Enterprise Applikationen. J2EE-basierte Applikationen setzen auf dieser Plattform auf und werden modular, komponentenbasiert und gegen einen festen Standard, die J2EE-Spezifikation, entwickelt. J2EE hat sich zur bedeutesten Plattform für unternehmenskritische Systeme entwickelt.

Für die im Rahmen der vorliegenden Diplomarbeit entwickelte Anwendung, stellt die J2EE-Plattform die Kerntechnologie dar. Wegen der zentralen Bedeutung von J2EE sowie der Komplexität des Themas, wird diesem Kapitel besondere Aufmerksamkeit gewidmet. Dies ist vor allem notwendig um die in Kapitel 6 beschriebene Umsetzung der Anwendung zu verstehen.

Die Architektur der Java 2 Enterprise Edition hat in der Vergangenheit eine große Beachtung auf dem Markt gefunden. Enterprise-Lösungen, welche das Ziel haben bestehende Legacy-Applikationen zu erweitern oder neue Applikationen zu entwickeln, um die Vorteile einer verteilen Architektur nutzen wollen, finden in der J2EE-Architektur eine Plattform, die die gestellten Anforderungen abdecken kann.

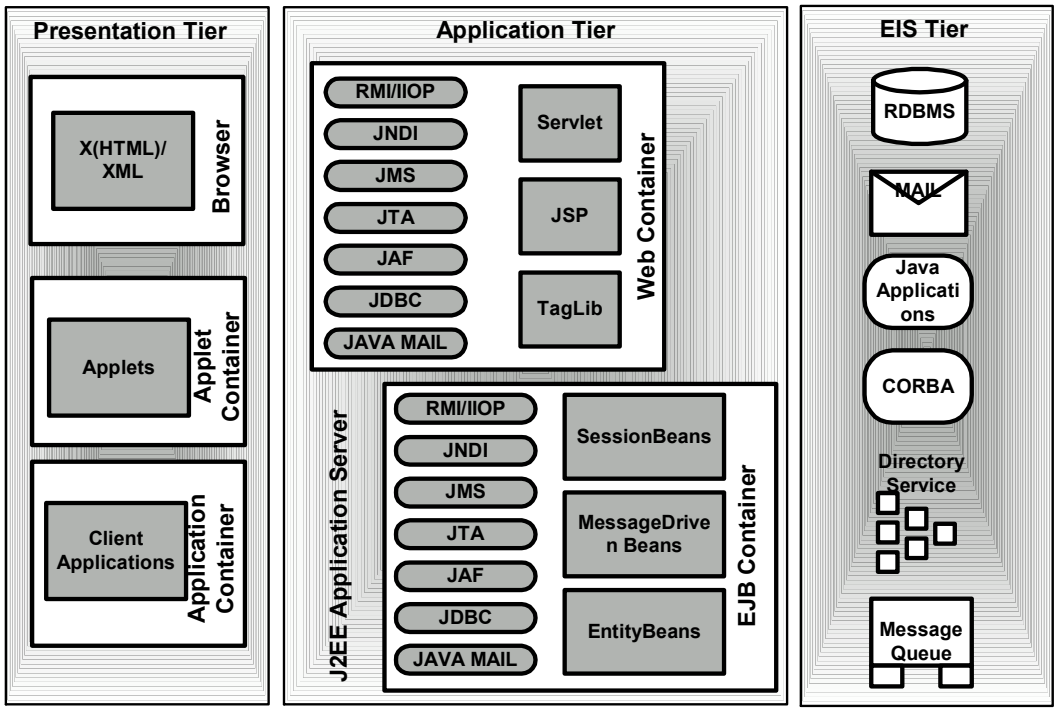

Abbildung 12 - Die J2EE-Architektur

Die J2EE-Plattform beinhaltet, neben der im nächsten Kapitel beschriebenen EJB-Spezifikation, eine Vielzahl verschiedener Dienste und Technologien. Dazu gehören insbesondere

- das Java Naming and Directory Interface (JNDI), welches den Zugriff auf Naming-Dienste wie DNS oder LDAP ermöglicht,
- die Remote Method Invocation (RMI) für den transparenten Zugriff auf entfernte Objekte,
- der Java Messaging Service (JMS), der die Unterstützung für eine asynchrone Kommunikation durch verschiedene Messaging-Systeme möglich macht,
- sowie die Java Interface Definition Language (IDL), die eine CORBA-Schnittstelle für Java zur Verfügung stellt.

5.1.5.2 Enterprise JavaBeans (EJBs)

Enterprise Java Beans sind plattformneutrale, wiederverwendbare Komponenten, die serverseitig ausgeführt werden und die Business-Logik einer Applikation implementieren[22]. Gemäß der Komponenten-Idee implementieren sie ein Menge von wohldefinierten Schnittstellen und können als Bausteine angesehen werden um ein größeres Problem leichter lösen zu können. Ein Unternehmen kann eine solche Komponente kaufen, sie mit anderen kombinieren und so auf einfache Art und Weise ein völlig neues Produkt entstehen lassen. Das wirklich Neue an Enterprise JavaBeans ist jedoch, dass die Entwicklung von großen verteilten Anwendungen erleichtert wird. Die Argumente dafür sind im Folgenden aufgeführt:

- Ein EJB-Container steuert die Beans und versorgt sie mit wichtigen System-Level-Services[23]. Entwickler können sich daher auf die Lösung der eigentlichen Geschäftsprobleme konzentrieren.
- Enterprise Java Beans erlauben eine deskriptive Programmierung. Dies bedeutet, dass eine Vielzahl von Einstellungen nicht „hart“ im Code verankert ist, sondern über einen „Deployment Descriptor“ spezifiziert wird. In Folge dessen erhöht sich die Portierbarkeit der Beans.
- Da Enterprise Java Beans portable Komponenten sind, können daraus leicht neue Applikationen erstellt werden, die auf verschiedenen Plattformen und in verschiedenen Servern einsetzbar sind.
- Die Enterprise Java Beans kapseln die gesamte Geschäftslogik einer Anwendung, wodurch die Entwicklung von „schlanken“ Clients für weniger leistungsstarke Geräte möglich wird.

[22] vergleiche Stefan Schäffer, Enterprise Java mit IBM Websphere, Seite 333
[23] System Level Services = Systemdienste wie z.B. Transaktionsmanagement

Eine Enterprise Java Bean stellt ein komplexes Konstrukt dar, das aus verschiedenen Komponenten besteht und in einer speziellen Umgebung laufen muss. Abbildung 13 illustriert diesen Zusammenhang.

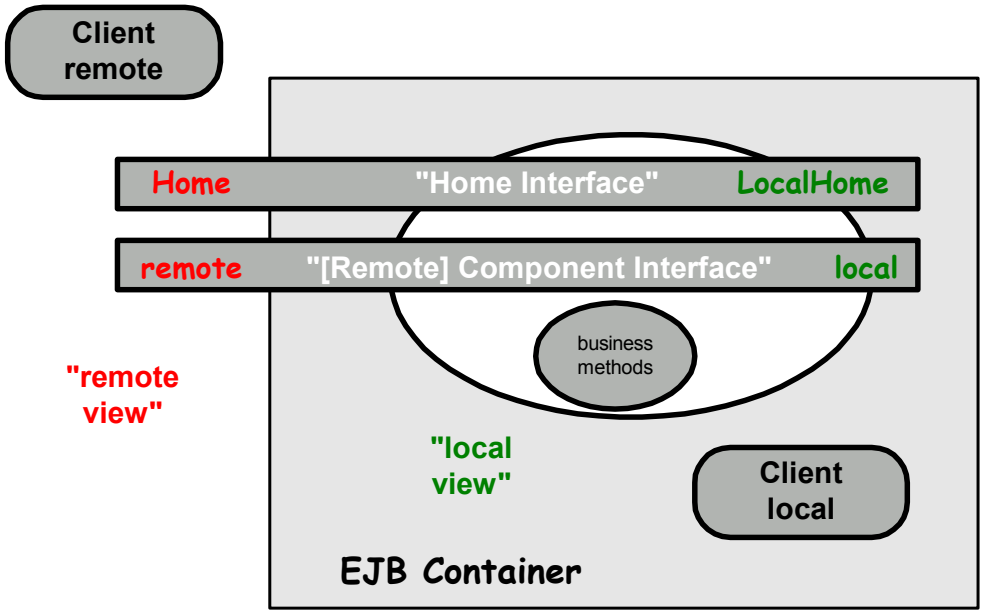

Abbildung 13 - Aufbau einer Enterprise Java Bean

Wie der Abbildung zu entnehmen ist und eingangs bereits erwähnt wurde, laufen EJBs in einer speziellen RunTime-Umgebung, dem EJB-Container. Der Container stellt die Schnittstelle zwischen einer Komponente und der sie unterstützenden umgebenden plattformspezifischen Funktionalität dar. Er fungiert als Dienstleister für die immer gleichen Anforderungen der Business Objekte. Am Container lassen sich über einen „Deployment Descriptor“ Einstellungen vornehmen, welche die Unterstützung durch den J2EE Server gewährleisten. Das betrifft z.B. die Sicherheit, das Transaktionsmanagement sowie die Client-Verbindung. Container sorgen daneben noch für Dienste, die nicht konfigurierbar sind, wie beispielsweise das Management des Lebenszyklus von EJBs und Servlets, das Pooling von Datenbankverbindungen, die Datenpersistenz und den Zugang zu anderen APIs der J2EE-Plattform.

Abbildung 13 zeigt weiterhin, dass eine EJB über zwei Schnittstellen (Interfaces) verfügt. Das Home-Interface bildet die Schnittstelle, die für den Lebenszyklus einer EJB verantwortlich ist. Es stellt Methoden zum Erzeugen, Löschen und Auffinden einer EJB bereit. Das (Remote) Component Interface hingegen definiert die Schnittstelle zu den Geschäftsfunktionen, die von einer EJB nach außen angeboten werden. Die Schnittstellen können außerdem als „Remote“ oder „Local“ deklariert werden. Als „Local“ deklarierte Interfaces können nur von Objekten aufgerufen werden, die sich im selben Container wie die EJB befinden. Anders ist dies bei Remote-Interfaces, die auch von außerhalb des Containers liegenden Objekten aufgerufen werden können, allerdings weniger performant sind.

Die Enterprise Bean Klasse sorgt für die eigentliche Implementierung der Geschäftsmethoden. Ruft ein Client eine Methode am Component Interface auf, so wird vom Container die entsprechende Geschäftsmethode in der Enterprise Bean Klasse angesprochen.

Es existieren folgenden Ausprägungen von EJBs:

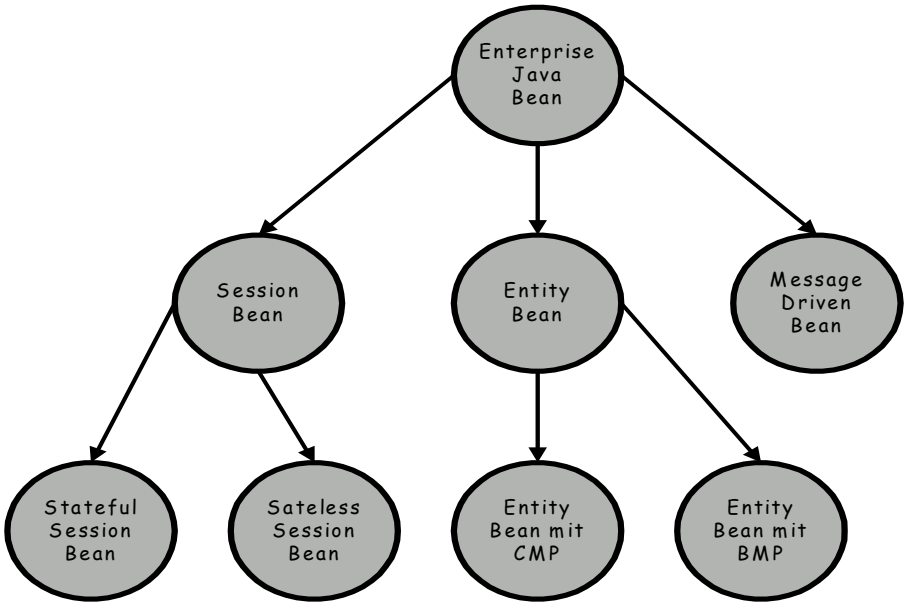

Abbildung 14 - Arten von Enterprise Java Beans[24]

[24] Stefan Schäffer, Enterprise Java mit IBM Websphere, Seite 334

- Session Beans

  Eine Session Bean modelliert die Abläufe und Vorgänge eines Prozesses und beinhaltet somit die Geschäftslogik einer Anwendung. Sie repräsentiert genau einen Client im J2EE-Server. Der Client ruft die Methoden einer Session Bean auf, welche dann Aufgaben für den Client ausführt und ihn so von der Komplexität der verschiedenen Business Tasks abschirmt. Eine Session Bean ist nicht persistent. Dies bedeutet, dass die Session Information nicht in einer Datenbank gespeichert werden, sondern spätestens nach dem Sitzungsende verloren gehen.

  Man unterscheidet zwei Arten von Session Beans: *Zustandbehaftet* (*Stateful*) und *Zustandslos* (*Stateless*). Der Status einer Stateful Session Bean wird durch die Instanzvariabeln einer Klasse repräsentiert und bleibt während der gesamten Sitzung vorhanden. Da man behaupten kann der Client spreche mit der Bean wird dieser Zustand auch als *conversational state* bezeichnet. Bei einer Stateless Session Bean hingegen werden bei jedem Methodenaufruf die Werte der Instanzvariabeln neu belegt. Stateless Session Beans können so während ihres Lebenszyklussees von mehreren Clients verwendet werden und sind daher performanter.

- Entity Beans

  Eine Entity Bean repräsentiert ein Business-Objekt in einem persistenten Speicher. Dies kann beispielsweise ein Kunden oder Produkt innerhalb einer Datenbank sein. Jede Entity Bean besitzt einen Primärschlüssel mit dem ein Client diese lokalisieren kann. Im Gegensatz zu Session Beans erlauben Entity Beans einen gemeinsamen Zugriff (Shared Access) und können Beziehungen zu anderen Entity Beans haben.

  Bezüglich der Datenspeicherung unterscheidet man zwischen *container-managed Persistence(CMP)* und *bean-managed Persistence(BMP)*. Beim BMP-Verfahren implementiert die jeweilige Bean den Code für sämtliche

Datenbankzugriffe. Der Mehrwert dieser Variante liegt in der Flexibilität und Mächtigkeit der selbstdefinierten SQL-Anweisungen. Bei der CMP-Variante werden hingegen alle Zugriffe auf die Datenbank zentral vom EJB-Container gesteuert. Die Unabhängigkeit des zugrundliegenden Datenbanksystems sowie die Reduzierung des zu implementierenden Codes sind hier die klaren Vorteile.

- MessageDriven Beans

Nachrichtengesteuerte Beans (MessageDriven Beans) sind zustandslose, transaktionsorientierte Komponenten zur Verarbeitung asynchroner JMS-Nachrichten. Eine der bedeutesten Eigenschaften von MessageDriven Beans (MDB) ist das nebenläufige Empfangen und Verarbeiten von Nachrichten. Der EJB Container verwaltet die Nebenläufigkeit automatisch, so dass sich der Entwickler auf die Geschäftslogik zur Verarbeitung der Nachrichten konzentrieren kann. Im Container können mehrere Instanzen einer MDB gleichzeitig ausgeführt werden, wodurch die Bean eine Vielzahl von JMS-Nachrichten empfangen und alle gleichzeitig verarbeiten kann.

	Session Beans	Entity Beans	MessageDriven Beans
Zweck	Führen eine Aufgabe für eine Client aus.	Repräsentieren ein Geschäftsobjekt in einem persistenten Speicher.	Arbeiteten als Listener für die Java Message Service API bei der Verarbeitung asynchroner Nachrichten
Shared Access	Nur ein Client	Kann von vielen Clients genutzt werden	Kann von vielen Clients genutzt werden
Persistence	Nicht persistent	Persistent	Nicht persistent

Tabelle 1 - Eigenschaften von Enterprise Java Beans

5.1.6 Java Message Service (JMS)

5.1.6.1 Message Oriented Middleware

Eine der in Kapitel 4.2 definierten funktionalen Anforderungen an den Netzsimulator ist die Fähigkeit, die verschiedenen Komponenten in einem heterogenen Netzwerk auf unterschiedlichen Systemen verteilen zu können. Das Stück Software, das den verteilten Komponenten einer Anwendung zu kommunizieren ermöglicht, bezeichnet man als Message Oriented Middleware. Dabei unterscheidet man zwei Arten von Kommunikationsformen. Die erste Form ist die synchrone, fest gekoppelte Kommunikation von verteilten Komponenten, auf der Technologien wie RMI, CORBA und DCOM basieren. Bei der zweiten Variante handelt es sich um eine asynchrone, lose gekoppelte Kommunikation, die man als Messaging bezeichnet. Im Netzsimulator kommen beide Varianten zum Einsatz. In diesem Kapitel soll jedoch zunächst die zweite Form, das Messaging, beschrieben werden.

Das charakteristische am Messaging ist, dass „one-way“ Nachrichten versendet werden, die keine sofortige Antwort vom Kommunikationspartner erwarten. Der Sender der Nachricht ist daher nicht blockiert und kann sich zwischenzeitlich anderen Aufgaben widmen. Sender und Empfänger kommunizieren nicht direkt miteinander, sondern ausschließlich über einen Vermittler (Message Server).

JMS ist eine standardisierte, herstellerunabhängige API, die eine Kommunikation mit verschiedenen MOM-Systemen ermöglicht. JMS kann analog zu JDBC betrachtet werden. Während JDBC einen herstellerneutralen Zugriff auf verschiedene, relationale Datenbanksysteme erlaubt, stellt JMS den gleichen unabhängigen Zugriff auf unterschiedliche Nachrichtensystem zur Verfügung.

5.1.6.2 JMS Architektur

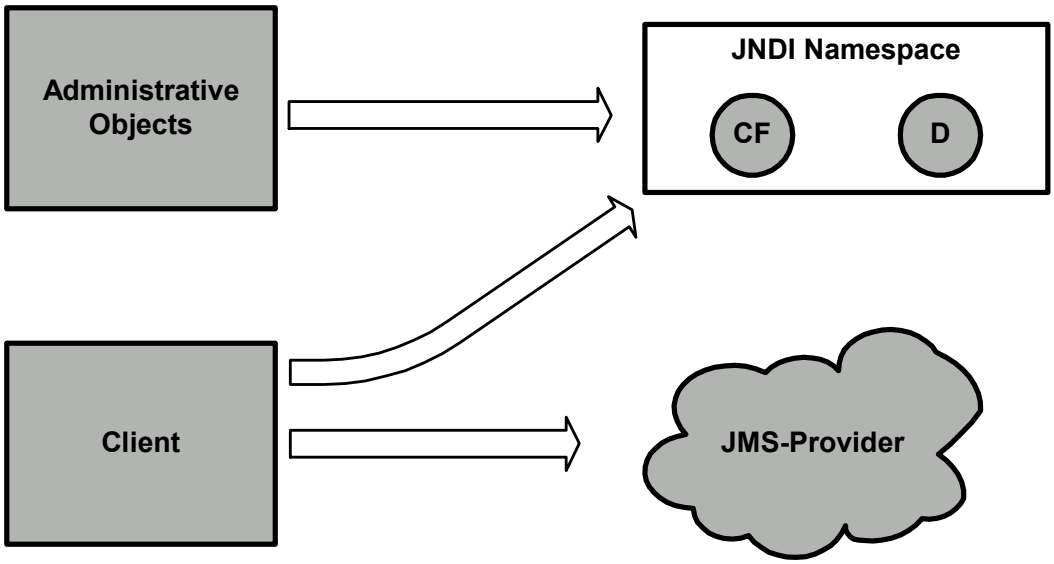

Abbildung 15 - Komponenten der JMS-Architektur

Die JMS-Architektur besteht aus den folgenden Parteien:

- Der *JMS-Provider* ist ein Nachrichtensystem, welches das JMS-Interface implementiert und administrative sowie steuerungstechnische Methoden bereitstellt. Er ist im wesentlichen für die Weiterleitung der Nachrichten zuständig und kann mit einem Hub verglichen werden, mit dem alle Clients verbunden sind.
- *JMS-Clients* sind in Java implementierte Programme oder Komponenten, die Nachrichten produzieren und/oder konsumieren.
- *Messages* sind die Informationsträger für die Kommunikation zwischen den JMS-Clients.
- *Administrated Objects* sind vorkonfigurierte JMS-Objekte die von einem Administrator für die Clients erzeugt wurden. Es gibt zwei Arten von *administrated Objects*, *Destinations* und *Connection Factories,* welche weiter unten beschrieben werden.

5.1.6.3 Nachrichtenmodelle (Messaging Domains)

In JMS existieren zwei verschiedene Kommunikationsmodelle[25]: *publish-and-subscribe* (Veröffentlichen und Abonnieren) und *point-to-point* (Punkt-zu-Punkt).

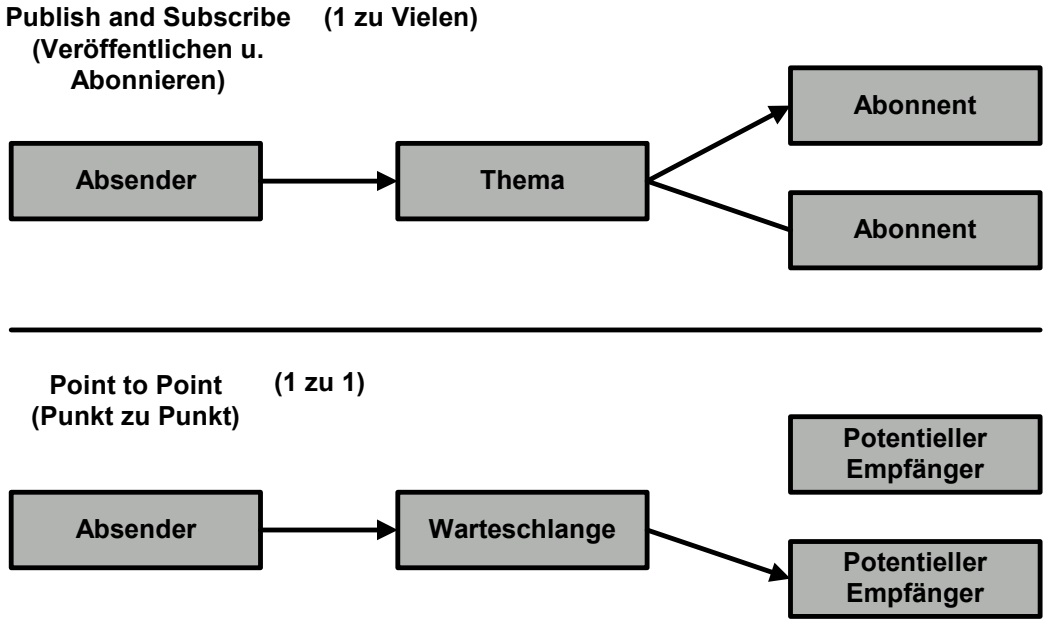

Abbildung 16 - JMS Nachrichtenmodell

Beim publish-and-subscribe (pub/sub) Verfahren schickt ein JMS-Client, der Produzent, eine Nachricht an ein Thema (Topic). Sämtliche Nachrichten, die an das Thema gesendet worden sind, werden den Empfängern des Themas als Kopie zugestellt. Die Empfänger müssen dabei nicht kontrollieren ob neue Nachrichten vorliegen, sondern bekommen diese automatisch vom Thema zugeschickt.

Im point-to-point (p2p) Nachrichtenmodell kann ein JMS-Client Nachrichten sowohl synchron wie auch asynchron über eine Warteschlange (Queue) empfangen und versenden. Im Gegensatz zum pub/sub Programmiermodell, wo eine unaufgeforderte Zustellung erfolgt, werden die Nachrichten hier explizit aus

[25] Vergleiche Professional Enterprise Java Beans S 226

der Warteschlange angefordert. Bezeichnend für das p2p Verfahren ist außerdem, dass für eine Nachricht zwar mehrere potentielle Empfänger vorhanden sein können, die Nachricht schließlich aber nur von einen Empfänger konsumiert werden kann.

5.1.7 Remote Method Invocation (RMI)

Nachdem im vorherigen Kapitel die JMS-Technologie erläutert wurde, befasst sich dieses Kapitel mit einer weiteren Form des *Distributed Object Computing*, der Remote Method Invocation (RMI). In der vorliegenden Arbeit wird RMI zum Austausch von Informationen zwischen den Komponenten des Netzsimulators verwendet und trägt damit der Anforderung, ein verteilbares System zu schaffen, Rechnung. RMI ist ein von Sun entwickelter RPC-Mechanismus, der vollständig auf Java basiert. Dies bedeutet das es sich sowohl beim Client wie auch beim Server um eine Java-Applikation handeln muss.

Im Gegensatz zu JMS erfolgt die Kommunikation nicht indirekt über einen Vermittler, sondern über direkte Methodenaufrufe eines entfernten Objektes. Bemerkenswert ist dabei, das für den Client dieser entfernte Methodenaufruf einem lokalen Aufruf gleicht. Der Client ist sich also nicht der Tatsache „bewusst“ mit einem Objekt zu kommunizieren, welches sich auf einem anderen System befindet. Ermöglicht wird diese transparente Kommunikation durch sogenannte Proxy-Objekte, welche sowohl auf der Server-Seite (Stub) wie auch auf der Client-Seite (Skeleton) existieren und stellvertretend für das entfernte Objekt stehen. Diese Proxy-Objekte implementieren die gleichen Schnittstellen wie das Remote-Objekt und haben im wesentlichen die Aufgabe, den tatsächlichen Übertragungsvorgang zu verdecken. Abbildung 17 illustriert diesen Zusammenhang.

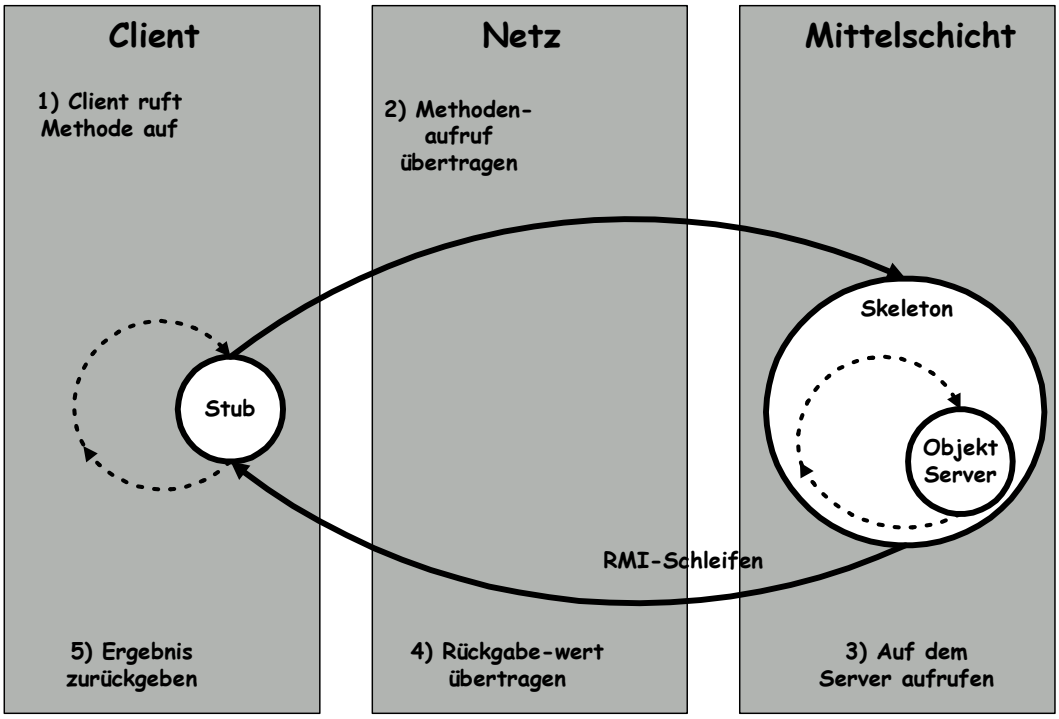

Abbildung 17 - RMI-Schleifen[26]

„EJB 2.0 und EJB 1.1 definieren die entfernten Schnittstellen einer Enterprise Java Bean mit „RMI over IIOP“ was für Corba[27]-Konformität sorgt. Dies bedeutet das das Protokoll, das von den entfernten Clients für den Zugriff auf Enterprise Java Beans verwendet wird, beliebig vom Hersteller gewählt werden kann.“[28]

5.1.8 Secure Socket Layer (SSL)

5.1.8.1 Der Sicherheitsaspekt

Eine der wichtigsten Anforderungen an den Netzsimulator ist die Berücksichtigung des Sicherheitsaspektes (vergleiche Kapitel 4.2). Eine Lösung, die man zur Realisierung dieser Anforderungen einsetzen kann, beruht auf dem

[26] Richard Manson, Enterprise Java Beans, S. 134
[27] **C**ommon **O**bject **R**equest **B**roker **A**rchitecture ermöglicht eine plattformunabhängige Kommunikation zwischen verschiedenen Anwendungen
[28] Richard Manson, Enterprise Java Beans, S. 134

SSL-Protokoll. SSL (Secure Socket Layer) ist ein von Netscape entwickeltes Sicherheitsprotokoll, welches der TCP/IP Protokollfamilie angehört und für die sichere Datenübertragung zwischen einem Client und einem Server entwickelt wurde. Erfahrungen aus der Vergangenheit haben gezeigt, dass es gerade beim Verbindungsaufbau mit SSL auf der VASP-Seite häufig zu Schwierigkeiten kommen kann. Aus diesem Grund ist es von zentraler Bedeutung die SSL-Schnittstelle im Netzwerksimulator nachzubilden.

5.1.8.2 Eigenschaften von SSL

SSL erweitert das TCP/IP Protokoll um zusätzliche Eigenschaften. Dazu gehören

- die **Verschlüsselung** zur Gewährleistung der Vertraulichkeit,
- die **Zertifizierung** zur Echtheitsbestätigung von Client und Server sowie
- der **Message Authentification Code** zur vollständigen und unverfälschten Nachrichtenübermittlung.

5.1.8.3 Technik der Verschlüsselung

SSL basiert auf den folgenden kryptongrafischen Techniken:

- symmetrische Verschlüsselung

  Bei der symmetrischen Verschlüsselung werden zur Ver- und Entschlüsselung die gleichen Schlüssel verwendet. Dies bietet durch den geringen Rechenaufwand den Vorteil, dass der Prozess relativ schnell durchgeführt werden kann. Der Nachteil dieser Methode liegt in der Problematik des Schlüsselaustauschs, da sowohl der Sender als auch der Empfänger über ein und denselben Schlüssel verfügen müssen.

- asymmetrische Verschlüsselung

Bei dem asymmetrischen Verfahren verwenden Sender und Empfänger unterschiedliche Schlüssel zur Ver- und Entschlüsselung. Bei diesen Schlüsseln handelt es sich um sogenannte Schlüsselpaare, die immer aus der Kombination von privatem und öffentlichem Schlüssel bestehen. Der öffentliche Schlüssel wird bekannt gegeben, während der private Schlüssel geheim bleibt. Eine Nachricht, die mit dem privatem Schlüssel codiert wurde, kann nur mit dem dazugehörenden öffentlichen Schlüssel wieder decodiert werden. Der Vorteil dieser Variante liegt in der einfachen Methode des Schlüsselaustauschs, da der öffentliche Schlüssel für jeden zugänglich ist. Nachteilig wirkt sich der hohe Rechenaufwand beim Ver- und Entschlüsseln aus.

- sichere Hash-Funktionen

  Eine Hash-Funktion wandelt einen Datenblock oder eine Nachricht mit variabler Länge in einen Wert mit fester Länge. Der Zweck dieser Funktion ist die Erstellung eines „Fingerabdrucks“ einer Nachricht. Es wird ein *Message Authentification Code* (MAC) erzeugt, welcher der Erhaltung der Datenintegrität dient. Der MAC-Wert wird aus einem bestimmten Teil jedes Datenpaketes erzeugt. Diese Verfahren sind so konstruiert, dass bereits eine Änderung von einem Bit der Eingabe ausreichen, um einen anderen Wert zu erzeugen. Bekannte Hashing-Algorithmen sind MD5 (Message Digest 5) und SHA-1 (Secure Hash Algorithsm).

SSL verwendet eine Kombination aus symmetrischen und asymmetrischen Verfahren. Daher findet man häufig auch die Bezeichnung hybrides Verfahren. Für die Übertragung der Nutzdaten wird ein symmetrischer *Session Key* verwendet, der sowohl dem Server als auch dem Client bekannt sein muss. Zur Übertragung des Session Keys vom Server an den Client kommt das asymmetrische Verfahren zum Einsatz.

Kapitel 6

Umsetzung

Nachdem in den vorhergehenden Kapiteln die wirtschaftlichen Aspekte beleuchtet, die Anwendungsidee beschrieben und die technischen Grundlagen vermittelt wurden, widmet sich dieses Kapitel dem Schwerpunkt der Arbeit. Dieser besteht in der Umsetzung des in Kapitel 4 beschriebenen Netzsimulators. Im ersten Abschnitt werden die Systemkomponenten und deren Anwendungszweck beschrieben. Im nächsten Abschnitt folgt dann die detaillierte technische Darlegung der Komponentenentwicklung innerhalb der einzelnen Anwendungsschichten. Der letzte Abschnitt stellt schließlich die verschiedenen Produkte, die in der Programmier- und Systemumgebung zum Einsatz kommen, vor.

6.1 Komponenten

6.1.1 Verteilung der Komponenten im System (Deployment)

Der Netzsimulator besteht aus drei elementaren Komponenten. Dazu gehören

- ein Steuerungsinterface zur Administration der Mobilstationen,
- der Wirknetz-Simulator zur Nachbildung des Mobilfunknetzes,
- sowie eine Web-Applikation zur Simulation eines Value Added Service Providers.

Das Zusammenspiel und die Verteilung der einzelnen Elemente im Schichtenmodell, wird durch das folgende Deployment-Diagramm deutlich.

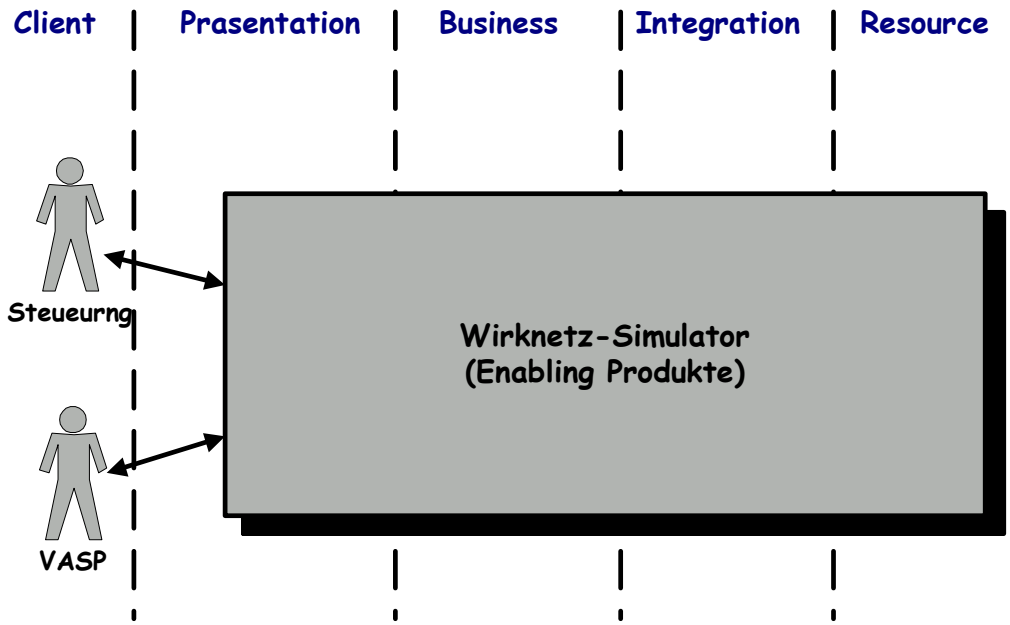

Abbildung 18 - Deployment der Komponenten des Netzsimulators

Das Steuerungsinterface und der VASP befinden sich in der Client-Tier. Die restlichen Schichten werden vom Wirknetz-Simulator eingenommen.

6.1.2 Das Steuerungsinterface

6.1.2.1 Anwendungszweck

Das Steuerungsinterface dient primär zur Simulation der Mobilstationen. Dabei werden die Funktionen und Eigenschaften einer Mobilstation abgebildet, die für die in Kapitel 4.3 identifizierten Anwendungsfälle erforderlich sind. Darüber hinaus ermöglicht das Steuerungsinterface ein Monitoring bestimmter Wirknetz-Prozesse.

6.1.2.2 Aufbau der Benutzeroberfläche

Beim Design der Benutzeroberfläche wurde versucht eine übersichtliche und intuitiv bedienbare Anwendung zu schaffen. Daraus ergab sich eine Aufteilung des User Interfaces in fünf Sektionen.

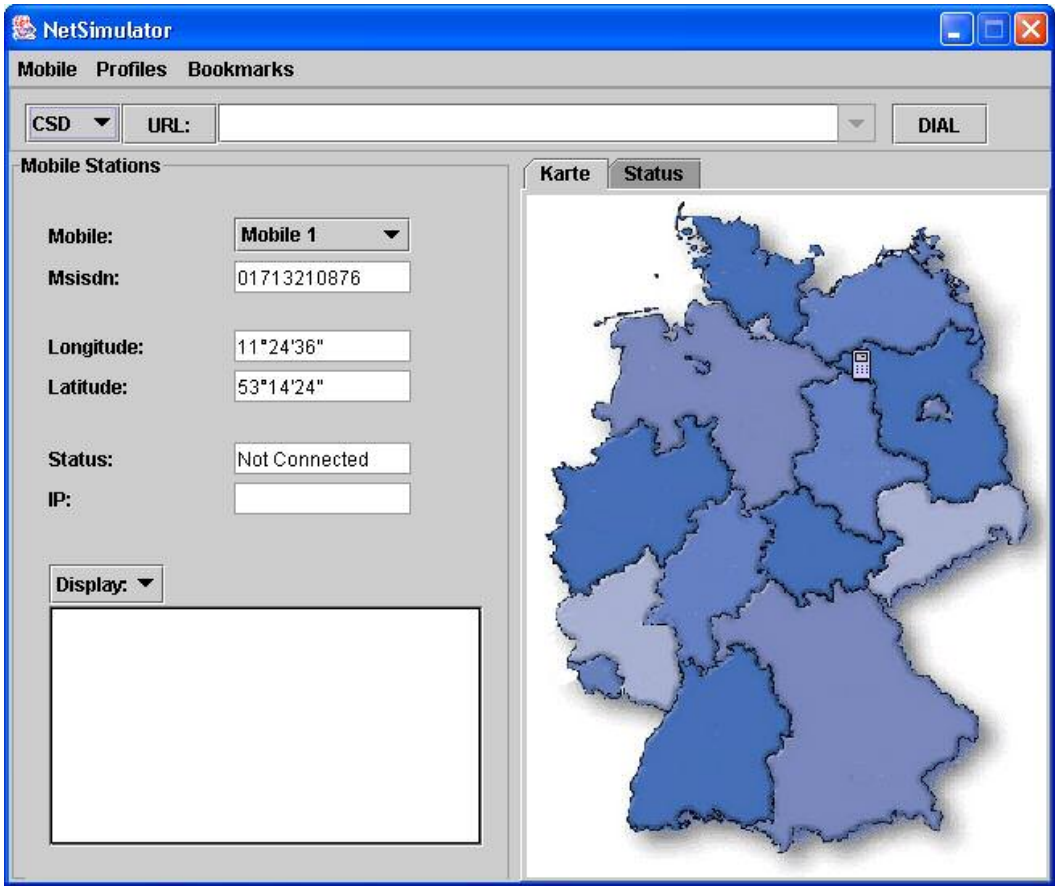

Abbildung 19 - Die Steuerungsoberfläche

I. Menüleiste

In der Menüleiste sind die administrativen Funktionen angesiedelt. Dazu gehören das Anlegen und Entfernen einzelner Mobilstationen, die Anzeige verfügbarer Profile sowie die Verwaltung der Bookmarks.

II. Verbindungsleiste

Die Verbindungsleiste ist vergleichbar mit der Adressleiste gängiger Browser (z.B. Internet Explorer). In der linken Auswahlbox kann zunächst die Verbindungsart bestimmt werden. Durch die Eingabe einer URL in das Textfeld wird die Internet-Adresse eines Value Added Service Provider spezifiziert. Über den rechten Button kann dann eine Verbindung auf- und wieder abgebaut werden.

III. Mobilstation

In der Sektion „Mobilstation“ finden sich alle wichtigen Eigenschaften einer Mobilstation wieder, die für die Simulation notwendig sind.

- *Name und MSISDN*

Der Name und die MSISDN werden beim Erzeugen einer Mobilstation automatisch generiert und dienen der Identifikation des Endgerätes innerhalb der Steuerungsoberfläche sowie in der Wirknetz-Emulation.

- *IP-Adresse*

Die IP-Adresse wird nach einem Verbindungsaufbau einer Mobilstation im Wirknetz erzeugt und von diesem zugewiesen.

- *Koordinaten*

Die Koordinaten entsprechen der Position des Endgerätesymbols auf der Karte.

- *Display*

Ganz unten aufgeführt ist das Display, welches vorwiegend zur Darstellung von Informationen dient. Dabei handelt es sich typischerweise um die Antwortseite einer Internet-Anfrage. Außerdem kann das Display

aber auch für die Eingabe von SMS-Texten verwendet werden.

IV. Karte

Ein wesentliches Charakteristikum von Mobilfunknetzen, ist das sich Mobilstationen innerhalb des Netzes bewegen. Diese Bewegung kann durch eine Verschiebung der Symbole auf der Karte simuliert werden.

V. Monitoring

Die Interaktionen zwischen dem Steuerungsinterface und dem Wirknetz, sowie zwischen dem Value Added Service Provider und dem Wirknetz werden in der Monitoring-Sektion dargestellt. Die Darstellung erfolgt in Form einer Baumstruktur, wodurch auch Teilprozesse abgebildet und Werte einzelner Parameter eingesehen werden können.

Hierdurch soll dem Value Added Service Provider die Fehleranalyse erleichtert werden. Neben dem Monitoring der eigenen Prozesse können auch die Aktionen einer dritten Person überwacht werden. Dies geschieht durch die Angabe einer VASP-ID in der unteren Auswahlbox.

6.1.3 Der Wirknetz-Emulator

6.1.3.1 Anwendungszweck

Der Wirknetz-Simulator bildet das zentrale Kernstück der Anwendung. Das Hauptziel dieser Emulation ist es die Schnittstellen und das externe Verhalten der Enabling Produkte nachzubilden. Aus Sicht des VASP müssen daher die Komponenten CRISP, LBS und CIA simuliert werden. Darüber hinaus werden aber auch die für die Einwahlprozesse der Mobilstationen verantwortlichen Komponenten nachgestellt.

6.1.3.2 Wirknetz-Elemente

Im Folgenden sind die wichtigsten Elemente der Wirknetz-Simulation dargestellt.

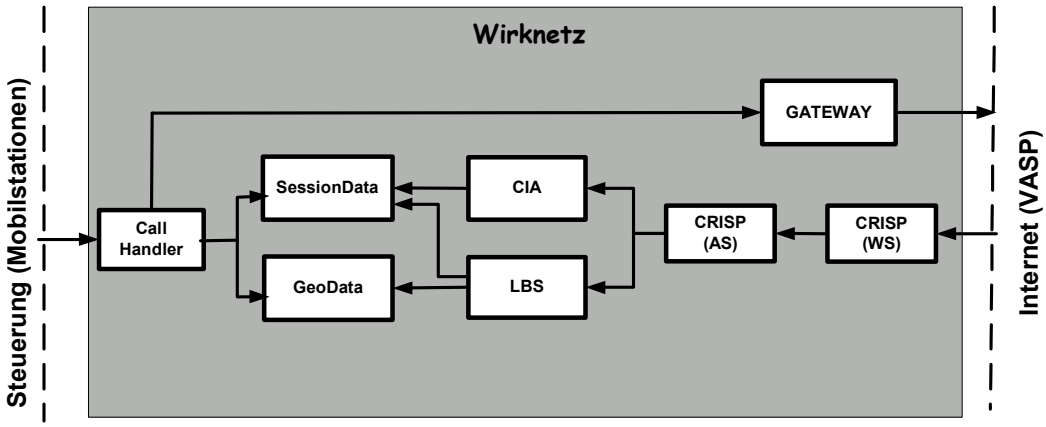

Abbildung 20 - Elemente des Wirknetzes

I. CRISP

Die CRISP-Komponente besteht aus zwei Klassen. Eine Klasse simuliert den CRISP-Webserver, die andere den CRISP-Applikationsserver. Aufgabe der Komponente ist die Authentifizierung und Autorisation des VASP sowie die Delegation der VASP-Anfragen an die Komponenten LBS oder CIA.

II. LBS

Die Komponente LBS repräsentiert den LBS-Server. Sie liefert Informationen bezüglich der Lokalität eines Kunden und dient der Auflösung einer Zell-ID in geographische Koordinaten.

III. CIA

Die Komponente CIA spiegelt den CIA-Server wider. Sie gibt Auskunft

über die Identität eines Mobilfunkkunden.

IV. CallHandler

Die Komponente CallHandler umfasst die Simulation des BSS und MSC. Sie ist verantwortlich für den Verbindungsaufbau und -abbau einer Mobilstation.

V. Gateway

Die Gateway Komponente repräsentiert das WAP-Gateway. Sie leitet die Anfragen einer Mobilstation in das Internet weiter.

VI. SessionData

In dieser Komponente werden die Einwahlinformationen einer Mobilstation gespeichert.

VII. GeoData

Dieses Element enthält Informationen bezüglich der Zellstruktur, in die sich das simulierte Datennetz aufteilt.

6.1.4 VASP Applikation

6.1.4.1 Anwendungszweck

Ein Value Added Service Provider (VASP) wird in Form einer Web-Applikation simuliert. Zur Bereitstellung von standortbezogenen Informationen (Location Based Services) implementiert die Anwendung einen Wetterinformations-Service. Die VASP-Applikation ist eine eigenständige Anwendung, die nicht

direkt zum Netzsimulator gehört. Ziel der Anwendung ist es, einen VASP Beispielapplikation zur Demonstrationszwecken sowie zur Vervollständigung des Gesamtbildes zu realisieren. Darüber hinaus ist es denkbar das Teile dieser Applikation später von VASPs als Module in eigenen Anwendungen genutzt werden können.

6.1.4.2 Funktionen der VASP-Applikation

Die VASP-Applikation stellt eine webbasierte Benutzerschnittstelle für die Nutzung der Wirknetz-Dienste „Location Based Services“, „Network Info Request“ und „Customer Identity Application“ zur Verfügung. Für jeden Dienst können eine Vielzahl von Parametern eingestellt werden. Die meisten dieser Werte haben einen konstanten Charakter und können über ein Formular persistent gespeichert werden. Es existieren aber auch Parameter, die sich für jeden Aufruf ändern. Dazu gehört z.B. die Zell-ID in einem NetworkInfo-Request. Diese Argumente finden sich daher in der abgebildeten Dienste-Maske wieder und können so vor jedem Aufruf neu gesetzt werden.

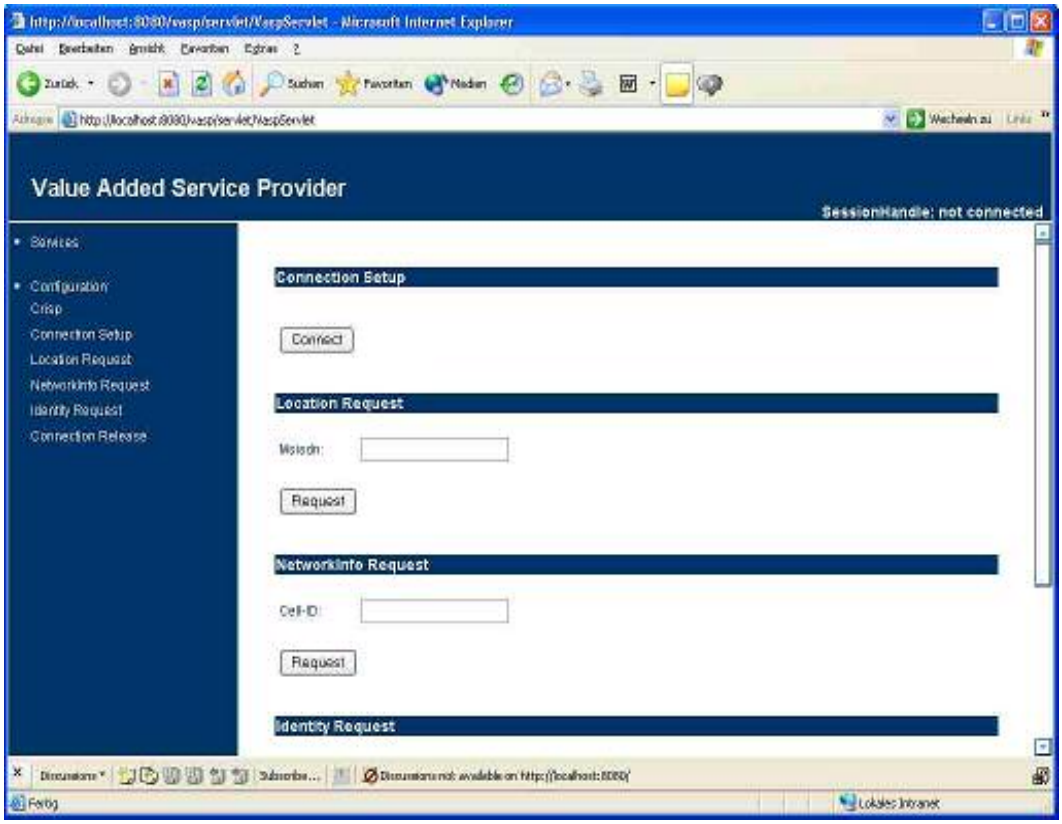

Abbildung 21 - Die VASP-Applikation

- Connection Setup

  Bevor ein Dienst genutzt werden kann, muss zunächst über die Funktion „Connection Setup“ eine Verbindung aufgebaut werden. Als Resultat erhält der VASP-Client eine Session-ID, über die er sich bei der Nutzung weiterer Dienste identifiziert.

- Location Based Services (LBS)

  Der LBS-Dienst liefert Informationen bezüglich der Position einer Mobilstation. Als Identifikation für die Mobilstation kann eine Rufnummer oder IP-Adresse in das Textfeld „MSISDN“ eingeben werden.

- NetworkInfo Request

Der NetworkInfo Request ermittelt für eine Zell-ID die zugehörigen Koordinaten. Die Zell-ID kann in das entsprechende Textfeld eingegeben werden.

- Customer Identity Application (CIA)

  Der CIA-Dienst liefert zu einer IP-Adresse die entsprechende Rufnummer der Mobilstation zurück.

- Connection Release

  Nach der Nutzung eines Dienstes kann die Session-ID zurückgegeben und die Verbindung zum Wirknetz-Simulator beendet werden.

6.3 Architektur

6.3.1 Systemüberblick

Die Abbildung 22 zeigt die Systemarchitektur der Anwendung. Sie gliedert sich in die in Kapitel 5.1 vorgestellten Anwendungsschichten. Der Aufbau der einzelnen Schichten und Komponenten wird in den folgenden Abschnitten detailliert erläutert.

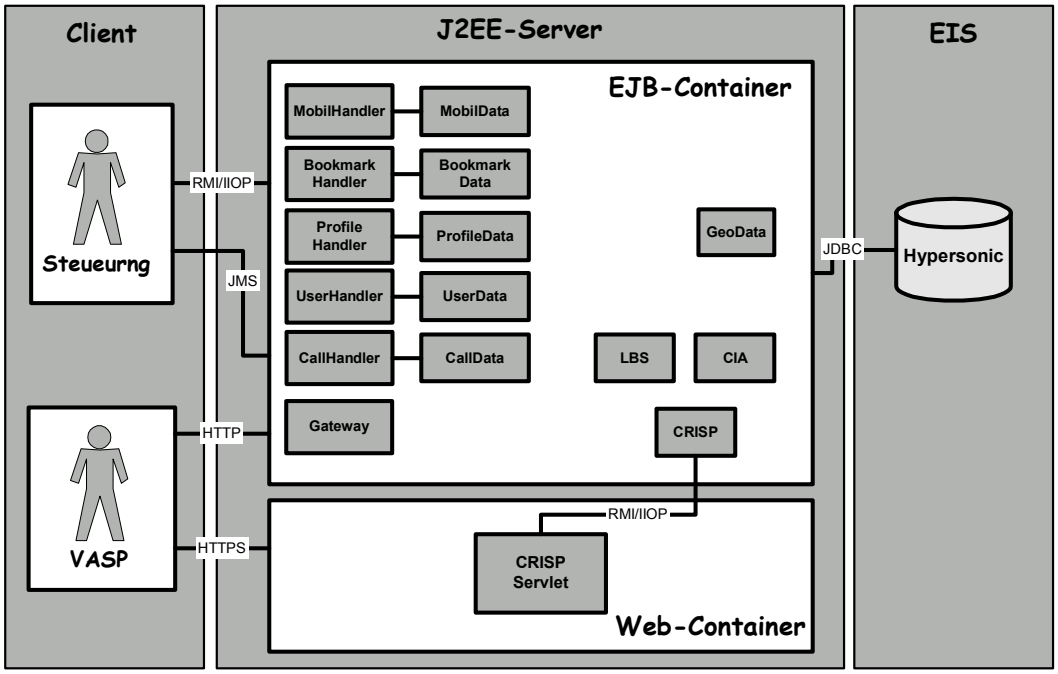

Abbildung 22 - Architektur des Netzsimulators

6.3.2 Client-Tier

In dieser Schicht sind zwei Arten von Clients vorzufinden: Das Steuerungsinterface und der Value Added Service Provider. Obwohl beide Clients auf die Wirknetz-Simulation zu greifen, unterscheiden sie sich stark hinsichtlich

der Zugriffsart. Sie verwenden unterschiedliche Protokolle und sprechen verschiedene Schichten an.

6.3.2.1 Das Steuerungsinterface

Wie bereits in Abschnitt 6.1.2 beschrieben dient das Steuerungsinterface zur Simulation der Mobilstationen. Der Client ist als Java-Applikation realisiert. Die Oberfläche wurde mit Hilfe der Java-Swing-Technologie entwickelt. Sie setzt sich aus den in Kapitel 6.1.2 beschriebenen Komponenten zusammen. Diese Komponenten bezeichnet man unter Swing als Panels. Panels sind Container, die andere Elemente (z.B. Buttons oder Textfelder) enthalten können und einen bestimmten Funktionsbereich abgrenzen. So existiert beispielsweise ein Panel für das Menü und ein weiteres für die Mobilstationen. Die Kommunikation zwischen den einzelnen Panels erfolgt gemäß dem Mediator-Pattern. Dem Pattern zu folge tauschen sich die Panels nicht direkt aus, sondern kommunizieren ausschließlich über einen Mediator. Dies hat den Vorteil, dass keine Abhängigkeit zwischen den einzelnen Panels besteht, da diese lediglich den Mediator kennen. Durch diese lose Koppelung betreffen Änderungen an der Schnittstelle eines Panels nur den Mediator. Abbildung 23 illustriert diesen Zusammenhang.

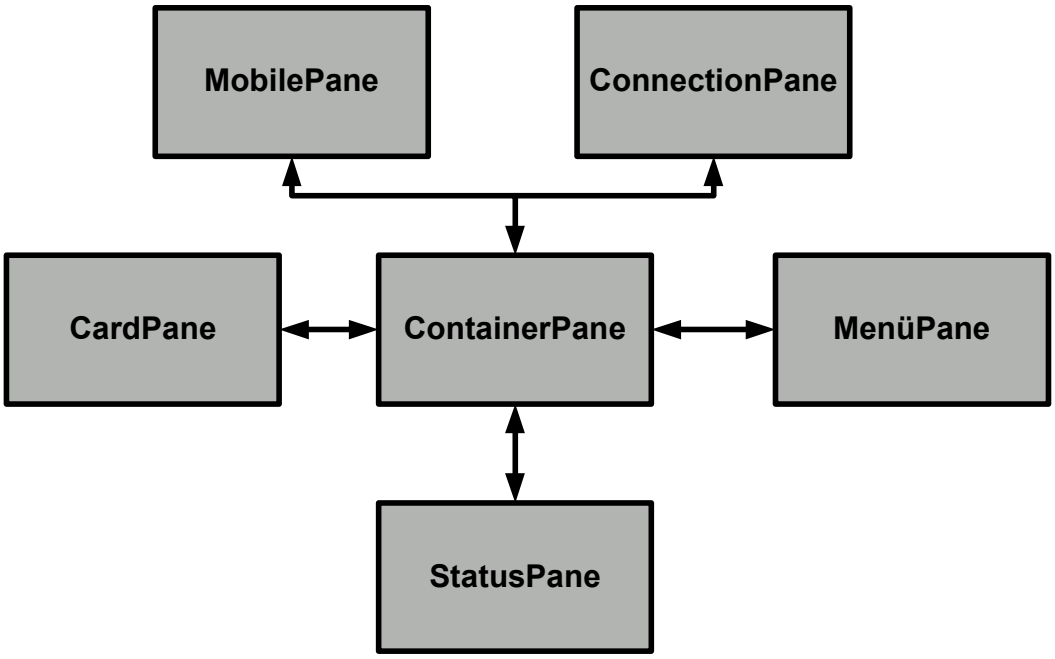

Abbildung 23 - Mediatorpattern in der Steuerungsoberfläche

Das Steuerungsinterface greift direkt auf die Business-Tier der Wirknetz-Simulation zu. Ein Zugriff auf die Präsentations-Tier erfolgt nicht, da sich die Präsentationslogik hier innerhalb des Clients befindet. Die Kommunikation zwischen einer Mobilstation und dem Wirknetz erfolgt über JMS. Dies umfasst den Verbindungsaufbau und –abbau, das Anfragen einer Internet-Seite oder das Versenden einer SMS. In einem MOM-System wie JMS werden die Informationen nicht direkt zwischen den kommunizierenden Objekten ausgetauscht, sondern an eine „Destination[29]“ übermittelt. Das Steuerungsinterface sendet daher zunächst die Daten an ein Topic. Das Topic delegiert die Nachricht dann weiter an die jeweilige Wirknetz-Komponente. Durch diese asynchrone Kommunikation, müssen Steuerungsinterface und Wirknetz die Daten nicht zur gleichen Zeit verarbeiten. Hierdurch ist es möglich nach dem Ausführen einer Funktion (z.B. Anfragen einer Internet-Seite) im Steuerungsinterface weiterzuarbeiten. Darüber hinaus können durch den „Public and Subscribe“ Mechanismus des Topics mehrere Empfänger eine Nachricht konsumieren. Letzteres ist insbesondere für die Multi-Client-Fähigkeit des Systems von zentraler Bedeutung.

[29] Destination = Topic oder Queue. Siehe Kapitel 5.1.6.3

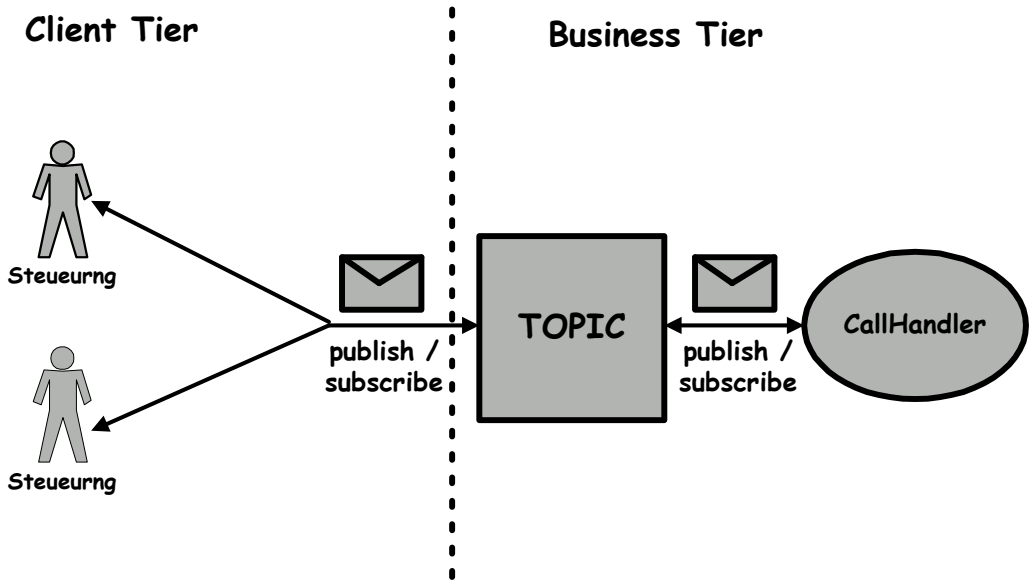

Abbildung 24 - Zugriff der Steuerungsoberfläche auf die Business-Tier

Für die administrativen Funktionen der Steuerungsoberfläche kommt das Protokoll „RMI over IOOP“ zum Einsatz. Zu diesen Funktionen gehören beispielsweise das Anlegen einer Mobilstation oder die Anzeige der verfügbaren Profile. Bei „RMI over IIOP“ werden direkt die Methoden der jeweiligen Wirknetz-Objekte aufgerufen. Wie bereits in Kapitel 5.1.7 beschrieben erfolgt die Kommunikation über die Proxy-Objekte, Stub und Skeleton. Abbildung 25 veranschaulicht den Datenaustausch zwischen Steuerungsinterface und Wirknetz.

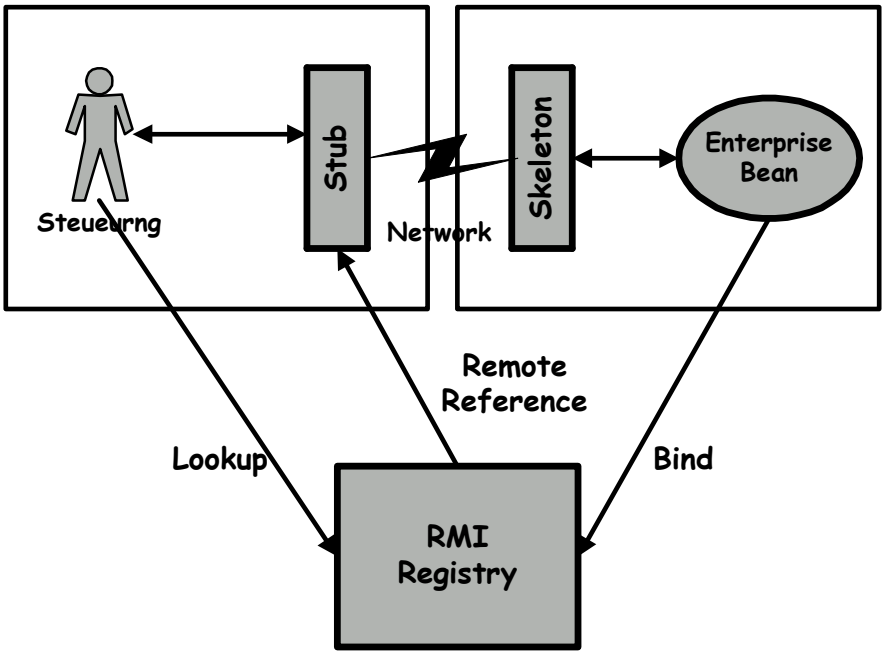

Abbildung 25 - Zugriff der Steuerungsoberfläche auf die Business-Tier

Die „direkte“ Kommunikation über RMI bietet Performancevorteile gegenüber JMS. Hinzu kommt das für die administrativen Funktionen die Vorteile von JMS - die asynchrone und „One-To-Many“-Kommunikation - keine Bedeutung haben.

6.3.2.2 Die VASP-Applikation

Die VASP-Applikation ist neben dem Steuerungsinterface in der Client-Tier angesiedelt. Die hier entwickelte Anwendung besteht ebenfalls aus einer Multi-Tier Architektur. Sie wurde nach dem MVC[30]-Pattern entwickelt. Das Pattern führt eine strikte Trennung zwischen Präsentation (*View*) und Anwendungsobjekten (*Modell*) ein. Die Reaktionen auf Benutzereingaben werden vom *Controller* implementiert. Durch das MVC-Pattern wird eine deutlich erhöhte Flexibilität und Wiederverwendbarkeit geschaffen. So können View-Elemente ausgetauscht oder angepasst werden, während die eigentliche Applikation, das Model, unverändert bleibt. Genauso können Änderungen in der Algorithmik des Models durchgeführt werden, ohne das die Notwendigkeit

[30] **M**odel **V**iew **C**ontroller

besteht, die Präsentation oder die Anwendungsobjekte anzupassen. Die Abbildung 26 stellt die Beziehungen zwischen Model, View und Controller der VASP-Applikation dar.

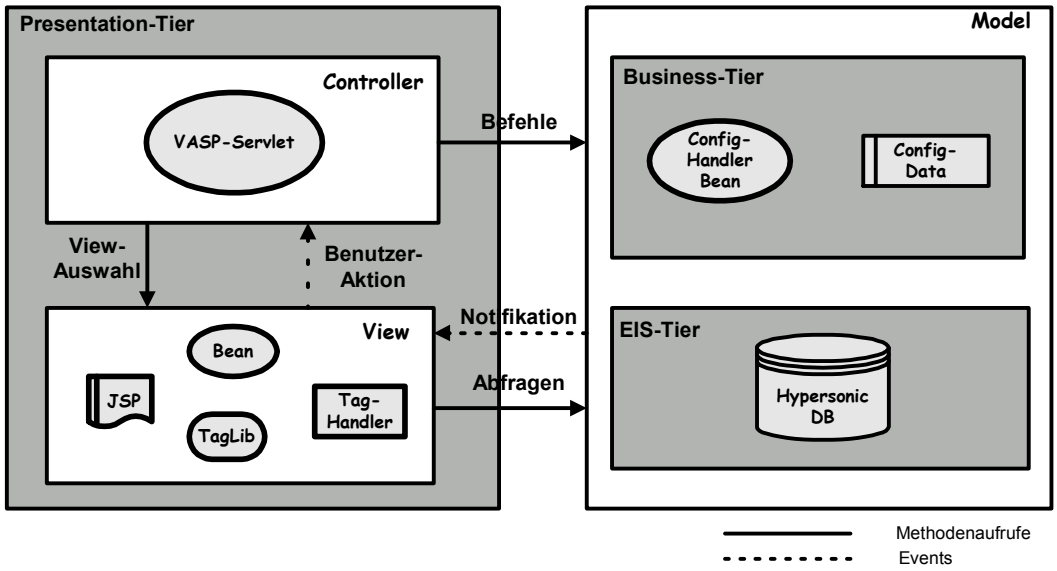

Abbildung 26 - Aufbau der VASP-Applikation nach dem MVC-Pattern

Die JSP-Seiten, welche die View repräsentieren, leiten die Benutzeraktionen direkt an das VASP-Servlet (Controller) weiter. Hier ist die Antwortstrategie implementiert, die festlegt, wie Eingaben zu verarbeiten sind. Aufgrund der Benutzeraktionen kann das VASP-Servlet einen anderes JSP-File selektieren und Befehle an den ConfigHandler (Model) schicken. Die JSP-Files werden vom ConfigHandler über Zustandsänderungen informiert, woraufhin sie die geänderten Werte des Models abfragen können. Für die Visualisierung der Ergebnisdaten werden in den JSP-Seiten *View Helper* eingesetzt. View Helper sind Tag Bibliotheken bzw. JavaBeans, welche die Ausgabe der Ergebnisdaten entsprechend dem gewünschten Layout anzeigen.

Während das Steuerungsinterface mit den Protokollen „RMI over IIOP“ und „JMS“ direkt die Business-Tier der Wirknetz-Simulation anspricht, greift der VASP mit dem Protokoll „HTTP over SSL“ auf die Präsentations-Tier zu. Durch diese gesicherte Verbindung wird die Authentizität der Parteien sowie die Vertraulichkeit der Daten garantiert. Sowohl der VASP wie auch das Wirknetz

authentifizieren sich durch ein Zertifikat. Nach dem Aushandeln der SSL-Verbindung, dem sogenannten SSL-Handshake, beginnt der eigentliche Datenaustausch.

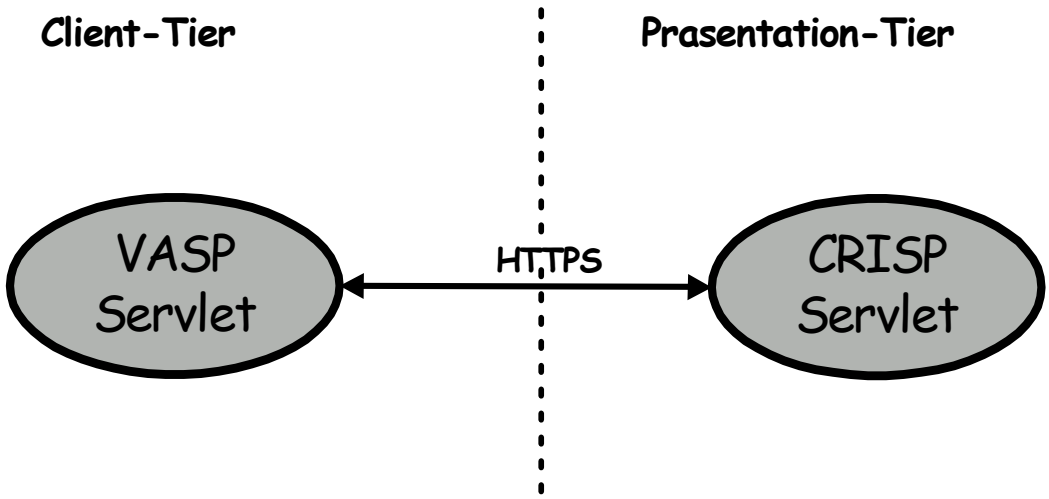

Abbildung 27 - Zugriff des VASP auf die Präsentations-Schicht

Als Antwort erhält die VASP-Applikation die Nutzerinformationen. Dies können beispielsweise bei einem Location Request, die Positionsdaten einer Mobilstation sein. In diesem Fall ermittelt die Anwendung dann die gewünschten Wetterinformationen und sendet sie der Mobilstation zu.

6.3.3 Presentation-Tier

In der Präsentations-Tier befindet sich das CRISP-Servlet. Es steht stellvertretend für den in Kapitel 3.1 beschriebenen CRISP -Webserver und stellt die zentrale Schnittstelle zum VASP-Client dar. Das Servlet ist gemäß dem Front Controller Pattern implementiert. Als Front Controller nimmt es alle Client-Anfragen entgegen und delegiert diese intern weiter. Aufgrund dieses zentralen Ansatzes wird die Präsentationslogik von der Geschäftslogik entkoppelt.

Wie bereits im vorhergehenden Kapitel beschrieben, kann auf die Präsentationsschicht nur über das Protokoll „HTTP over SSL“ zugegriffen

werden. Abbildung 28 zeigt das Handshake-Verfahren zwischen VASP und CRISP.

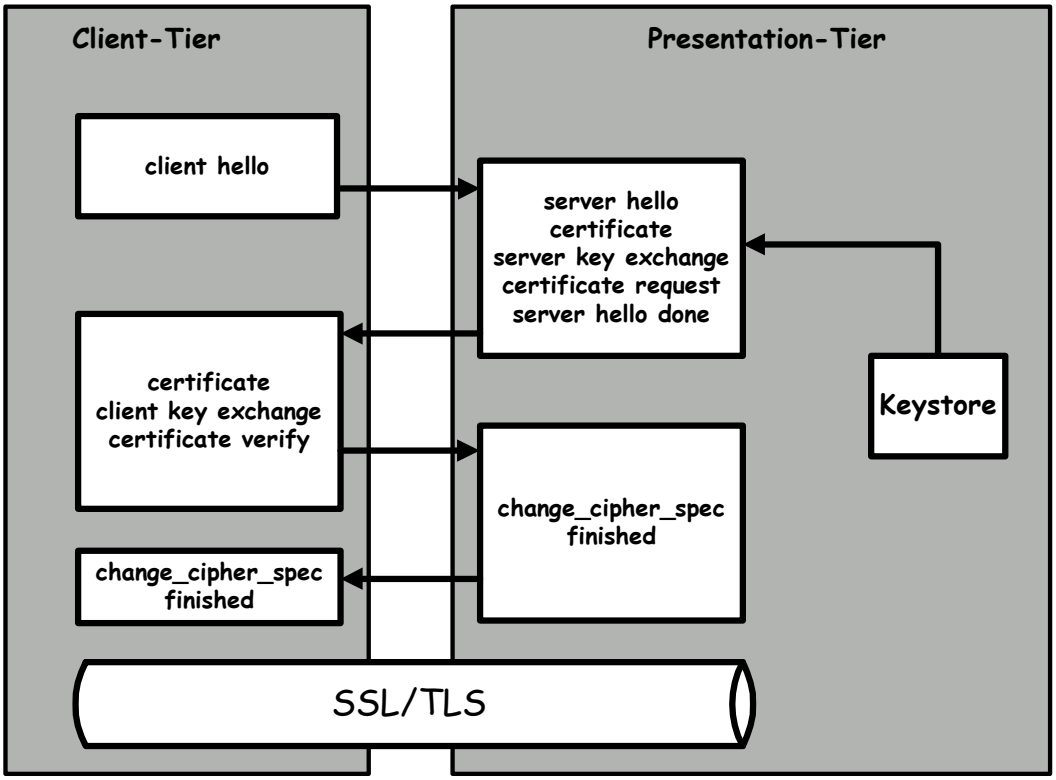

Abbildung 28 - SSL/TLS[31]-Handshake zwischen VASP und CRISP[32]

„Das Handshake-Protokoll ist dafür zuständig die Parameter des aktuellen Sitzungsstatus einzurichten“[33]. Zunächst werden die Sicherheitsfähigkeiten, wie z.B. Version und Kompressionsmethode, zwischen dem VASP und dem CRISP-Servlet ausgehandelt. Im nächsten Schritt authentifiziert sich der Server über ein elektronisches Zertifikat, übergibt seinen öffentlichen Schlüssel (public key) und fordert gleichzeitig ein Zertifikat zur Authentifizierung des Clients an. Letzterer erzeugt daraufhin einen Session-Key, mit dem später die eigentlichen Nutzdaten verschlüsselt werden. Der Session-Key wird vom Client mit dem „public key“ des Servers codiert und zusammen mit dem angeforderten Zertifikat an den Server gesendet. Im letzten Schritt werden die Listen bezüglich der

[31] TLS = Transport Layer Security. Nachfolger von SSL.
[32] Stephen Paine, Steve Burnett, Kryptographie, Seite 276
[33] Stephen Paine, Steve Burnett, Kryptographie, Seite 276

Verschlüsselungsalgorithmen und Hashgrößen aktualisiert und das Handshake-Verfahren beendet.

6.3.4 Business-Tier

6.3.4.1 Übersicht

Die Business-Tier kann als das Herzstück der Anwendung bezeichnet werden. Sie beinhaltet die Geschäftslogik und die Geschäftsdaten. Hier findet die eigentliche Verarbeitung der Benutzeranforderung sowie die Simulation der Enabling Produkte statt. Die folgende Abbildung zeigt den Aufbau der Schicht.

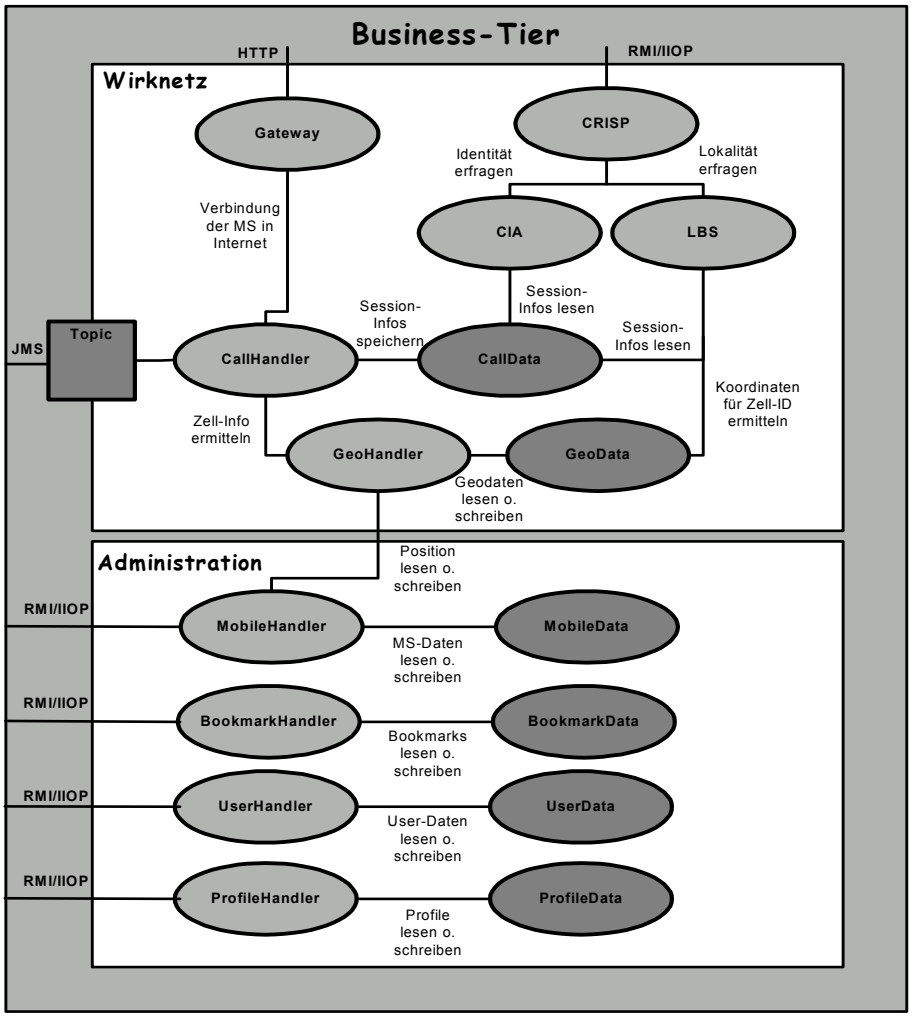

Abbildung 29 - Architektur der Business-Tier[34]

[34] siehe auch Klassendiagramme im Anhang D

Das Deployment-Diagramm zeigt alle Geschäftsobjekte der Business-Tier und stellt die Beziehungen zwischen diesen dar. Die hellblau gezeichneten Symbole repräsentieren die Session Beans, die dunkelblau gefärbten Ellipsen hingegen die Entity Beans. Während die Session Beans die Geschäftslogik implementieren, enthalten die Entity Beans die Geschäftsdaten. Darüber hinaus lässt das Diagramm eine Unterteilung der Business-Tier in zwei logische Bereiche erkennen. Die obere Sektion stellt die Simulation des Wirknetzes dar. Hier finden sich die bereits in Kapitel 6.1.3.2 beschriebenen Elemente wieder. Im unteren Bereich sind vorwiegend solche Objekte angesiedelt, die den administrativen Funktionen der Steuerungsoberfläche dienen.

Im Wirknetz sind drei Schnittstellen auszumachen. Die erste Schnittstelle, zum „CallHandler“, wird vom Steuerungsinterface für den Verbindungsaufbau und -abbau sowie für das Absenden einer Internetanfrage genutzt. Der Zugang erfolgt über das Protokoll „JMS“, wodurch eine asynchrone und multiple Kommunikation ermöglicht wird. Multiple bedeutet in diesem Zusammenhang, das eine Nachricht vom Wirknetz zu einer Mobilstation nicht nur vom initiierenden Client, sondern von mehreren Clients gesehen werden kann. Die zweite Schnittstelle bildet das Gateway. Anfragen einer Mobilstation werden von diesem ins Internet weitergeleitet. Über die dritte Schnittstelle sendet das CRISP-Servlet, welches sich in der Präsentationsschicht befindet, die VASP-Anfragen an die Business-Objekte weiter. Dabei wird für den Datenaustausch zwischen der Präsentationsschicht und der Business-Tier das Protokoll „RMI over IOOP“ eingesetzt.

Alle Zugriffe auf den administrativen Bereich der Business-Tier erfolgen ausschließlich über „RMI over IOOP“. Der Einsatz von JMS ist hier nicht sinnvoll, da die asynchrone Kommunikation an dieser Stelle keinen Mehrwert leistet und eine „One-To-Many“-Kommunikation nicht in Frage kommt. Darüber hinaus sind die RMI-Zugriffe performanter als bei JMS.

In den folgenden Kapiteln werden die Beschaffenheit und das Verhalten der einzelnen Komponenten beschrieben.

6.3.4.2 CallHandler[35]

Der CallHandler ist als Message-Driven Bean implementiert. Als solche verarbeitet er die Nachrichten (Messages) eines Topics. Bei diesen Nachrichten handelt es sich um Aktionen, die von einer im Steuerungsinterface befindlichen Mobilstation erzeugt wurden. Die folgenden Anwendungsfälle sind dabei möglich.

- Einwahl einer Mobilstation über CSD oder GPRS

  Verbindet sich eine Mobilstation über die Trägerdienste CSD oder GPRS, so wird zunächst anhand der übergebenen Koordinaten die Zell-ID ermittelt. Anschließend wird auf Basis eines Zufallsalgorithmussees eine IP-Adresse generiert. Zell-ID, IP-Adresse sowie die MSISDN werden als Session-Informationen der Mobilstation zwischengespeichert und stehen somit später anderen Diensten zur Verfügung. Schließlich wird die erzeugte IP-Adresse der Mobilstation zugewiesen. Dies geschieht durch das Versenden einer Nachricht an das Topic.

- Anfrage einer Internetseite von einer Mobilstation

  Fragt eine Mobilstation eine Internetseite an, so werden die erforderlichen Parameter „URL“ und „Client-IP-Adresse“ aus dem Nachrichten-Objekt extrahiert und an das Gateway weitergeben. Letzteres baut schließlich die Verbindung zum Internet auf.

- Versenden einer SMS

[35] siehe Klassendiagramm in Abbildung 47 in Anhang D

Versendet eine Mobilstation eine SMS, so wird anhand der übergebenen Koordinaten die Zell-ID ermittelt. Die Sender- und Empfänger-MSISDN werden zusammen mit der Zell-ID als Session-Informationen abgelegt. Schließlich wird die Nachricht übertragen und der Sender erhält eine Empfangsbestätigung.

6.3.4.3 Gateway[36]

Das Gateway ist als zustandslose Session Bean realisiert. Es nimmt die Anfrage einer Mobilstation vom CallHandler entgegen und formuliert daraus einen HTTP-Request. Das Gateway stellt somit die ausgehende Schnittstelle zum Internet dar. Das Ergebnis der Anfrage wird wiederum als Nachricht an die entsprechende Mobilstation im Steuerungsinterface gesendet.

6.3.4.4 Administrative Business-Objekte

Die im Folgenden beschriebenen Business-Objekte haben einen administrativen Charakter. Sie gehören nicht unmittelbar zur Simulation des Mobilfunknetzes, sondern erfüllen sekundäre Aufgaben. Damit sind beispielsweise die Bookmark-Verwaltung oder Profil-Ansicht gemeint. Um diese Funktionen abbilden zu können werden jeweils zwei Enterprise Beans implementiert. Zum einen handelt es sich dabei um eine zustandsbehaftete Session Bean. Die Klasse repräsentiert genau einen Client im J2EE-Server und bewahrt diesen vor der Komplexität der Business-Logik. Durch die Zustandsbehaftung der Session Bean bleiben die Client-Daten während der gesamten Sitzung erhalten. Bei der anderen Enterprise Bean handelt es sich um eine Entity Bean. Sie repräsentiert ein Objekt in einem persistenten Speicher. Der Zugriff auf die Entity Bean erfolgt ausschließlich über die Session Bean. Abbildung 30 illustriert diesen Zusammenhang.

[36] siehe Klassendiagramm in Abbildung 52 in Anhang D

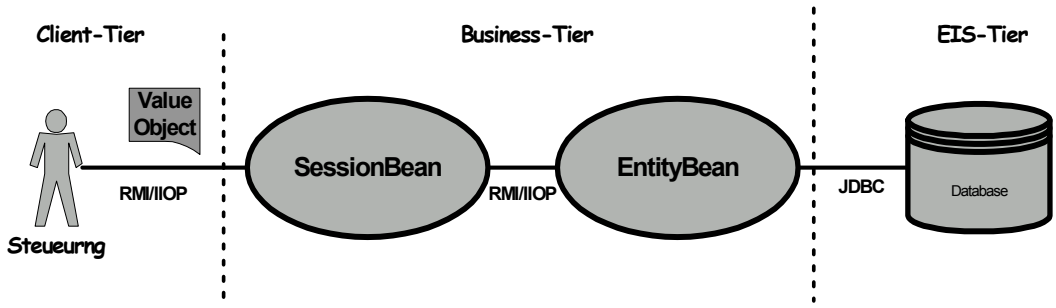

Abbildung 30 - Aufbau der administrativen Funktionen

Für den Austausch der Informationen zwischen der Client-Tier und der Business-Tier wird nach dem ValueObject Pattern verfahren. In einer EJB-Umgebung liegen Client- und Business-Tier gewöhnlich auf verschiedenen Servern. Dies bedeutet das es sich bei jedem Methodenaufruf um einen „teuren“ Remote-Zugriff handelt. Gemäß dem Pattern werden die Geschäftsdaten daher vom einen ValueObject gekapselt und können so in einem einzigen Zugriff transportiert werden.

An dieser Stelle werden kurz die administrativen Funktionen der Business-Tier beschrieben:

Administration der Bookmarks

Für jeden Benutzer wird eine eigene Bookmark-Liste gepflegt. Der BookmarkHandler[37] bildet als Session Bean die Schnittstelle zum Steuerungsinterface und stellt diesem Methoden zum Speichern und Abrufen einer Bookmark-Liste zur Verfügung. Das Schreiben und Lesen einzelner Bookmarks in bzw. von der Datenbank erfolgt über die Entity Bean „BookmarkData[38]“.

Verwaltung der Mobilstationen

[37] siehe Klassendiagramm in Abbildung 45 in Anhang D
[38] siehe Klassendiagramm in Abbildung 44 in Anhang D

Der MobileHandler[39] stellt für das Steuerungsinterface folgende Funktionalitäten bereit:

- Anlegen einer neuen Mobilstation.
- Löschen einer vorhandenen Mobilstation.
- Abrufen aller vorhandenen Mobilstationen.

Der Schreib- und Lesezugriff auf die Datenbank erfolgt über die Entity Bean „MobileData[40]“.

Ansicht der Service-Profile

Eine Liste aller verfügbaren Service-Profile kann über die Session Bean „ProfileHandler[41]“ vom Client abgerufen werden. Die Entity Bean „ProfileBean[42]“ liest die einzelnen Profile aus der Datenbank.

Benutzerverwaltung

Beim Starten des Steuerungsinterfaces wird der Benutzer aufgefordert einen Usernamen und ein Passwort einzugeben. Die Authentifizierung der Login-Informationen erfolgt innerhalb der Business-Tier durch die Session Bean „UserHandler[43]“. Jedem Benutzer wird beim ersten Login ein VASP-ID zugeordnet. Diese Nummer entspricht der ID, durch die sich der Value Added Service Provider beim Connection-Setup identifiziert. Ein VASP kann also nur solch eine ID verwenden, die einem Benutzer im Steuerungsinterface zugeordnet wurde. Ferner dient die VASP-ID der Unterscheidung einzelner Clients in der Wirknetz-Simulation. Im Rahmen der Benutzerverwaltung stellt der UserHandler folgende Leistungsmerkmale bereit:

[39] siehe Klassendiagramm in Abbildung 57 in Anhang D
[40] siehe Klassendiagramm in Abbildung 58 in Anhang D
[41] siehe Klassendiagramm in Abbildung 61 in Anhang D
[42] siehe Klassendiagramm in Abbildung 60 in Anhang D
[43] siehe Klassendiagramm in Abbildung 63 in Anhang D

- Login und Logout eines Benutzers.
- Anlegen und Löschen einzelner Benutzer.
- Ändern des Passwortes für einen Benutzer.
- Ermittlung der VASP-ID für einen Benutzer.
- Bereitstellung aller vorhandenen VASP-IDs.

6.3.4.5 CRISP

Die CRISP[44] Entität repräsentiert den in Kapitel 3.1 beschriebenen CRISP Applikationsserver. Zu den wesentlichen Aufgaben der Klasse gehören die Autorisation des Client, die Syntaxprüfung der Parameter, das Fehlermanagement sowie das Delegieren der Anfragen an die Enabling Produkte.

Bei der Klasse handelt es sich um eine zustandsbehaftete Session Bean, die nach dem BusinessDelegate-Pattern entwickelt worden ist. Sie stellt gemäß dem Pattern eine clientseitige Schnittstelle zu den Geschäftsdiensten der Business-Tier dar. Als Client tritt hier das in der Präsentations-Tier liegende CRISP Servlet auf. Die CRISP Session Bean reduziert die Koppelung zwischen der Präsentations-Tier und den Enabling Produkten, indem Implementierungsdetails verborgen bleiben. Häufige Änderungen an der API der Geschäfts-Objekte ziehen nur eine Anpassung des BusinessDelegate nach sich. Client-Komponenten bleiben so von der Volatilität der Enabling Produkte verschont. Abbildung 31 zeigt ein stark vereinfachtes Aktivitätsdiagramm der CRISP Bean, in dem die Hauptaktivitäten dargestellt sind.

[44] siehe Klassendiagramm in Abbildung 50 in Anhang D

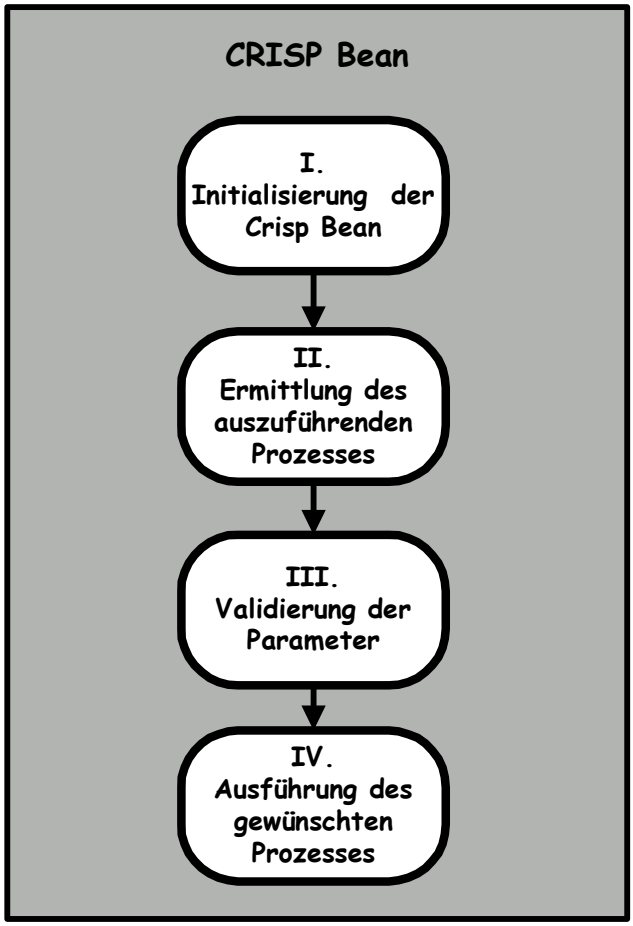

Abbildung 31 - Aktivitäten der CRISP Bean

I. Initialisierung der CRISP Bean

Die CRISP Session Bean wird vom CRISP Servlet, welches sich in der Präsentationsschicht befindet, instanziert und mit den Parametern der VASP-Anfrage aufgerufen.

II. Ermittlung des Prozesses

Zunächst überprüft die Bean anhand des Parameters „MESSAGE_TYPE“ um welchen der folgenden Prozesse es sich handelt.

- Connection Setup
- Location Request

- NetworkInfo Request
- Customer Identity Request
- Connection Release

III. Validierung der Parameter

Anschließend erfolgt eine Auswertung der übergebenen Parameter. Für die Validierung wird eine XML-Datei herangezogen, in der für jeden Parameter der Datentyp, der Wertebereich, die Verbindlichkeit sowie der Fehlercode spezifiziert sind.

Auszug aus der Datei "parameter.xml"

```
<CONNECTION_SETUP>
  <MESSAGE_TYPE type="Integer" range="0-255" class="M"
    error="MESSAGE_TYPE_INVALID"/>
  <VASP_ID type="Integer" range="0-65535" class="M"
    error="VASP_ID_INVALID"/>
  <REQUEST_ID type="Long" range="0-4294967296" class="M"
    error="REQUEST_TIME_INVALID"/>
  <PASSWORD type="String" range="[A-Za-z0-9]{5,10}" class="M"
    error="PASSWORD_INVALID"/>
</CONNECTION_SETUP>
```

Tabelle 2 - Die Datei "parameter.xml"

Die XML-Datei wird von einem SAX-Parser eingelesen und von einem Document-Handler zu einem Hashtable verarbeitet. Letzteres wird an die Klasse „ParameterEvaluator" übergeben, die schließlich die Gültigkeit der Parameter überprüft und gegebenenfalls entsprechenden Fehlermeldungen ausgibt.

I. Ausführung des Prozesses

Es existieren drei Arten von Anfragen, die von der CRISP Bean ausgeführt werden. Die nachfolgende Abbildung 32 stellte diese in der zeitlicher Reihenfolge dar. Der abgebildete *Enabling Produkt Request* fasst hier die drei Prozesse „Location Request", „NetworkInfo Request" und „Identity

Request“ zusammen.

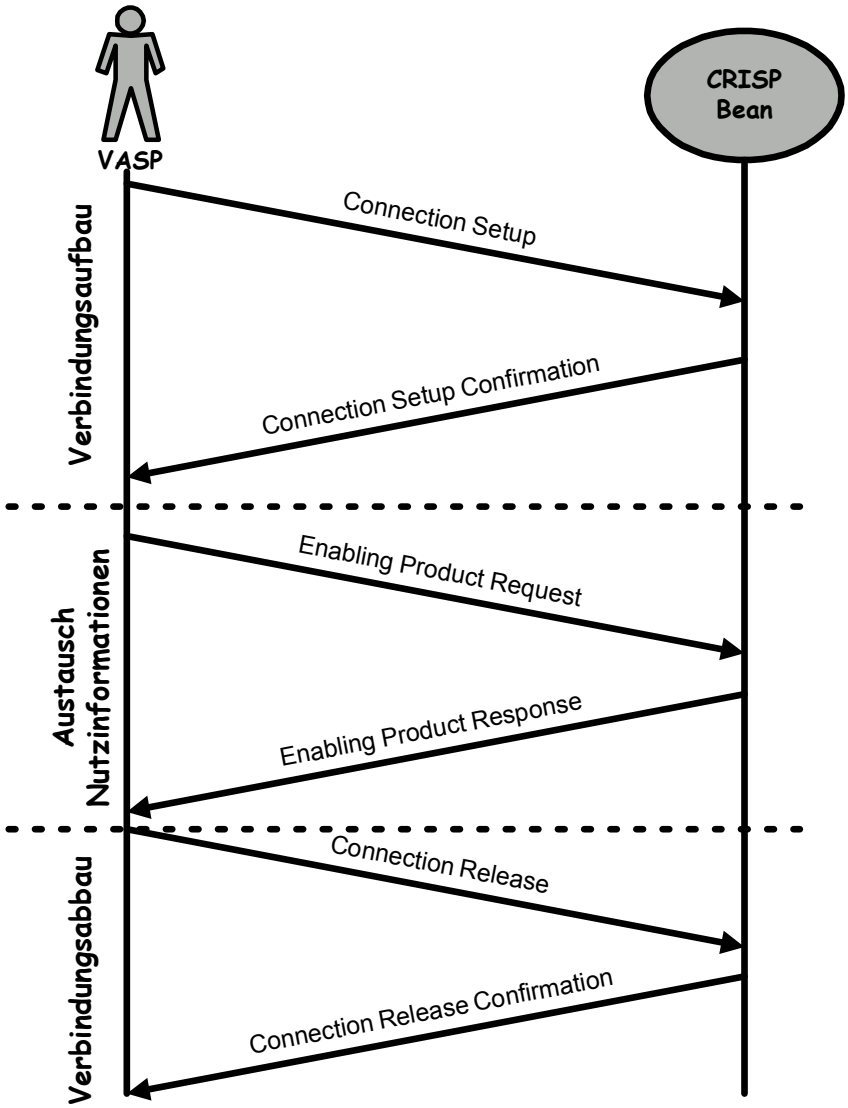

Abbildung 32 - Call-Flow der VASP-Anfragen

A. Connection Setup

Die Kommunikation zwischen dem Wirknetz und dem VASP wird durch die „Connection Setup“-Nachricht ausgehend vom VASP initiiert. Dabei übermittelt der VASP eine Request-ID, eine VASP-ID und ein VASP-Passwort. Die Parameter VASP-ID und Passwort werden bei der Einrichtung eines VASP-Profiles vergeben und dienen der Autorisierung.

Die Request-ID wird durch den VASP vergeben und dient ihm zur Unterscheidung verschiedener Anfragen, die er an den CRISP stellt.

VASP-ID und das Passwort werden zunächst überprüft. Nach erfolgreicher Autorisation wird dann ein Session-Handle generiert. Dieser zehnstellige numerische Wert wird in der CRISP-Bean zusammen mit der VASP-ID gespeichert. Der VASP gibt bei jeder weiteren Anfrage das Session-Handle mit, wodurch er einerseits ohne weitere Autorisation identifiziert wird, andererseits auf zwischengespeicherte Session-Information zugreifen kann.

Beispiel: Connection Setup Request
`<MESSAGE_TYPE=1><REQUEST_ID=1><VASP_ID=500><PASSWORD=testtest>`

Tabelle 3 - Connection Setup

Auf die Anfrage des VASP antwortet der CRISP mit der „Connection Confirmation“-Nachricht. Bestandteil der Antwort ist das generierte Session-Handle und die vom VASP erzeugte Request-ID, die ohne Modifikation an ihn zurückgegeben wird. Daneben finden sich noch zwei weitere Parameter. Die Status-Information beschreibt den Zustand der Verbindung. Treten Fehler im Zusammenhang mit der Angabe der Parameter auf (z.B. falsches Passwort), dann erscheint im Statusfeld ein Wert ungleich 0. Der Parameter „GMT-Difference“ gibt den Zeitunterschied des VASP zur Greenwich Mean Time an und zeigt so auch einen möglichen Zeitunterschied gegenüber dem CRISP auf. Dieser Parameter ist im weiteren Verlauf der Verbindung wichtig, wenn es um die Aktualität der Daten geht.

Beispiel: Connection Setup Confirmation
`<MESSAGE_TYPE=2><REQUEST_ID=2><SESSION_HANDLE=2854>` `<STATUS=0><GMT_DIFFERENCE=2>`

Tabelle 4 - Connection Setup Confirmation

B. Enabling Produkt Request

Nachdem die Verbindung zwischen den beiden Kommunikationspartnern aufgebaut ist, kann die eigentliche Abfrage stattfinden. Hierzu sendet der VASP eine der drei Anfragen: Location Request, NetworkInfo Request oder Customer Identity Request. Neben den allgemeinen Sitzungsparametern, die bereits im Vorfeld ausgetauscht wurden, enthält diese Meldung nun die eigentlichen Abfrageinformationen.

Anhand des vom Client übergebenen Session-Handles wird die VASP-ID ermittelt. Konnte diese gefunden werden, erfolgt die Instanzierung der jeweiligen Enabling Produkt Klasse und ein Aufruf der entsprechenden Methode. Gibt die Methode einen Fehlercode zurück, so wird dieser aufgelöst und eine entsprechende Fehlermeldung erzeugt. Andernfalls erfolgt die unveränderte Rückgabe der Antwort an den Client.

Beispiel: Error Response
`<MESSAGE_TYPE=13><REQUEST_ID=1><SESSION_HANDLE=2854>` `<STATUS=113>`

Tabelle 5 - Error Response

C. Connection Release

Durch die Nachricht „Connection Release“ wird im Anschluss an die Übertragung der Nutzinformationen der Verbindungsabbau eingeleitet. Das dabei vom Client übergebene Session-Handle wird verifiziert und alle damit verbunden Session-Information gelöscht.

Beispiel: Connection Release Request
`<MESSAGE_TYPE=3><REQUEST_ID=1><SESSION_HANDLE=2854>`

Tabelle 6 - Connection Release Request

Der VASP erhält als Bestätigung eine „Connection Release Confirmation“ Nachricht, welche die Sitzung zwischen dem CRISP und dem Client beendet.

Beispiel: Connection Release Confirmation
`<MESSAGE_TYPE=4><REQUEST_ID=1><SESSION_HANDLE=2854>` `<STATUS=0>`

Tabelle 7 - Connection Release Confirmation

6.3.4.6 LBS

Die LBS-Entität[45] repräsentiert den in Kapitel 3.1 beschriebenen LBS-Server. Während der CRISP für den Verbindungsaufbau und die Autorisation zuständig war, stellt die LBS-Klasse die eigentlichen Nutzinformationen, die von einem VASP angefragt werden, zur Verfügung. Dabei handelt es sich typischerweise um die Positionsdaten einer Mobilstation.

Die LBS-Klasse ist eine zustandslose Session Bean und beinhaltet als solche die komplette Geschäftslogik der bereitgestellten Dienste. Die Geschäftsdaten hingegen befinden sich in verschiedenen Entity Beans, auf die von der Session Bean zugegriffen wird. Zu den benötigten Daten gehören das Profil eines VASP, die Einwahlinformationen einer Mobilstation sowie geographische Informationen.

Von der LBS-Entität werden die Dienste „Location Request“ und „NetworkInfo Request“ implementiert, die im Folgenden detailliert beschrieben werden.

6.3.4.6.1 Location Request

[45] siehe Klassendiagramm in Abbildung 55 in Anhang D

Der Location Request[46] ermittelt anhand einer MSISDN oder IP-Adresse die Positionsdaten einer Mobilstation. Das Resultat des Dienstes kann vom VASP über eine Reihe verschiedener Parameter beeinflusst werden.

Beispiel: Location Request

```
<MESSAGE_TYPE=10><REQUEST_ID=1><SESSION_HANDLE=2854>
<SERVICE_ID=1><MAX_INFO_AGE=6000><MIN_GRANULARITY=2000>
<MOBILE_ID_TYPE=0><MOBILE_ID=491715556667>
<REQUEST_TIME=20021010><RESPONSE_DELAY=10000>
<GMT_DIFFERENCE=2>
```

Tabelle 8 - Location Request

Durch die Service ID wird das Format festgelegt, dass der VASP bei einer Anfrage erhält. Zu den unterstützten Formaten gehören beispielsweise „POINT“- oder „RECTANGLE. Weiterhin legt der VASP fest, wie alt die Nachricht über den Aufenthaltsort des Teilnehmers sein darf. Durch den Granularitäts-Parameter kann die gewünschte Genauigkeit der Nutzerposition (in Metern) angegeben werden. Je nach verwendetem Mobile-ID-Type (MSISDN oder dynamische IP-Adresse) muss im Parameter Mobile-ID die Rufnummer des Nutzers bzw. die IP-Adresse angegeben werden. Der Parameter „RESPONSE_DELAY“ bestimmt die maximale Bearbeitungszeit, die der Prozess in Anspruch nehmen darf.

Das folgende Aktivitätsdiagramm beschreibt die Ablaufmöglichkeiten eines Location Requests.

[46] siehe Sequenzdiagramm in Abbildung 64 in Anhang D

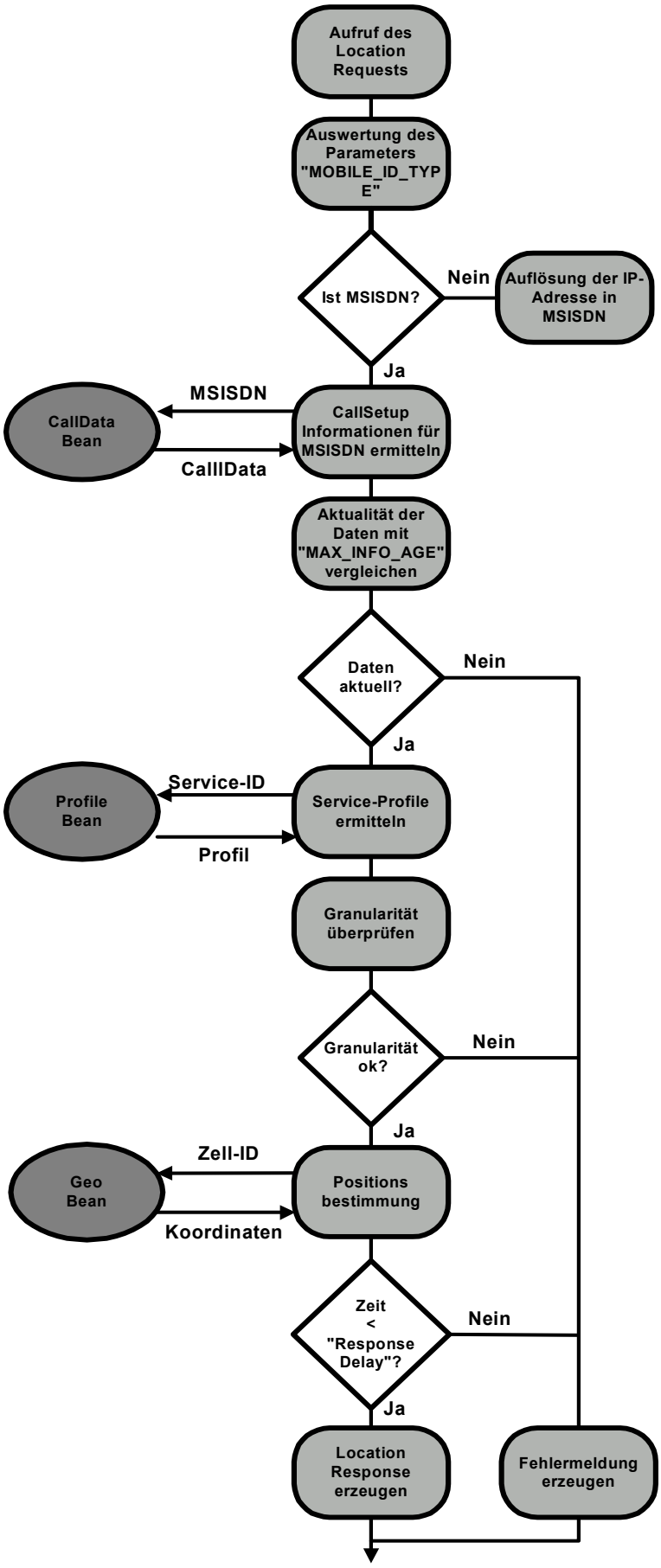
Aufruf des Location Requests
Auswertung des Parameters "MOBILE_ID_TYPE"
Ist MSISDN?
Nein
Auflösung der IP-Adresse in MSISDN
Ja
MSISDN
CallData Bean
CallIData
CallSetup Informationen für MSISDN ermitteln
Aktualität der Daten mit "MAX_INFO_AGE" vergleichen
Daten aktuell?
Nein
Ja
Service-ID
Profile Bean
Profil
Service-Profile ermitteln
Granularität überprüfen
Granularität ok?
Nein
Ja
Zell-ID
Geo Bean
Koordinaten
Positions bestimmung
Zeit < "Response Delay"?
Nein
Ja
Location Response erzeugen
Fehlermeldung erzeugen

Abbildung 33 - Aktivitätsdiagramm eines Location Requests

Zunächst wird der Parameter „MOBILE_ID_TYPE“ ausgewertet. Handelt es sich bei der Mobile-ID um eine IP-Adresse, so wird die entsprechende MSISDN aufgelöst. Mit Hilfe der MSISDN werden dann die Einwahl-Informationen der Mobilstation erfragt. Hierzu greift die LBS-Klasse auf die „CallData“-EntityBean zu, welche die erforderlichen Informationen kapselt. Da nun der Einwahlzeitpunkt der Mobilstation bekannt ist, wird anhand des Parameters „MAX_INFO_AGE“ überprüft ob die Daten der gewünschten Aktualität entsprechen. Anschließend wird über die Service-ID das Service-Profile ermittelt. Die Daten befinden sich in der „Profile“-EntityBean und geben im wesentlichen Auskunft über die Zugriffstechnologie (WAP oder SMS) sowie das zu verwendende Lokalisierungsformat (Cirlce, Point, Rectangle, usw.). Als nächstes gilt es die Granularität zu prüfen. Kann die gewünschte Genauigkeit gewährleistet werden, erfolgt die Positionsbestimmung der Mobilstation. Grundlage für deren Berechnung ist die in den Einwahl-Informationen enthaltene Zell-ID. In der „GEO“-Bean wird die Zell-ID in geographische Koordinaten umgerechnet, die dann innerhalb des spezifizierten Lokalisierungsformates an die LBS-Bean zurückgegeben werden. Konnten die einzelnen Prozedurschritte innerhalb der durch den Parameter „Response-Delay“ vorgegebenen Zeit abgearbeitet werden, erhält der VASP schließlich eine „Location Response“ Nachricht.

Beispiel: Location Response

```
<MESSAGE_TYPE=11><REQUEST_ID=1><STATUS=0><GEO_REF_SYSTEM=0>
<POSITION_FORMAT=2><MOBILE_ID_TYPE=0><MOBILE_ID=491715556667><
MOBILE_POSITION=<POSITION_FORMAT=CIRCLE;CENTER=555555,133333;R
ADIUS=646>><MOBILE_AGE_OF_INFO=2744><MOBILE_STATUS=4>
<TIMESTAMP=2002101012345>
```

Tabelle 9 - Location Response

6.3.4.6.2 NetworkInfo Request

Der NetworkInfo Request[47] ermittelt für eine Zell-ID die geographischen Koordinaten. Die meisten der dabei übergebenen Argumente sind schon aus dem Location Request bekannt.

Beispiel: NetworkInfo Request

```
<MESSAGE_TYPE=14><REQUEST_ID=1><SESSION_HANDLE=2854>
<SERVICE_ID=1><NETWORKINFO_ID_TYPE=0><REQUEST_TIME=20021010>
<RESPONSE_DELAY=10000><GMT_DIFFERENCE=2><MCC=49><MNC=12>
<LAC=221><CELLID=1001>
```

Tabelle 10 - NetworkInfo Request

Die Anfrage-Syntax sieht vor, dass neben der Zell-ID als Suchkriterium auch der „Location Area Code“ angegeben werden kann. Der Parameter „NETWORKINFO_ID_TYPE“ gibt in diesem Zusammenhang Auskunft darüber, nach welcher der beiden Größen die Berechnung der Koordinaten erfolgen soll. Der entsprechende Wert findet sich in den Parametern „CELLID“ oder „LAC“ wieder. Derzeit wird allerdings nur eine Auflösung nach der Zell-ID unterstützt. Über die Parameter „MCC“ (Mobile Country Code) und „MNC“ (Mobile Network Code) wird das Gebiet, auf das sich die Anfrage erstreckt, eingegrenzt.

Das folgende Aktivitätsdiagramm visualisiert die einzelnen Schritte im Verarbeitungsablauf.

[47] siehe Sequenzdiagramm in Abbildung 65 in Anhang D

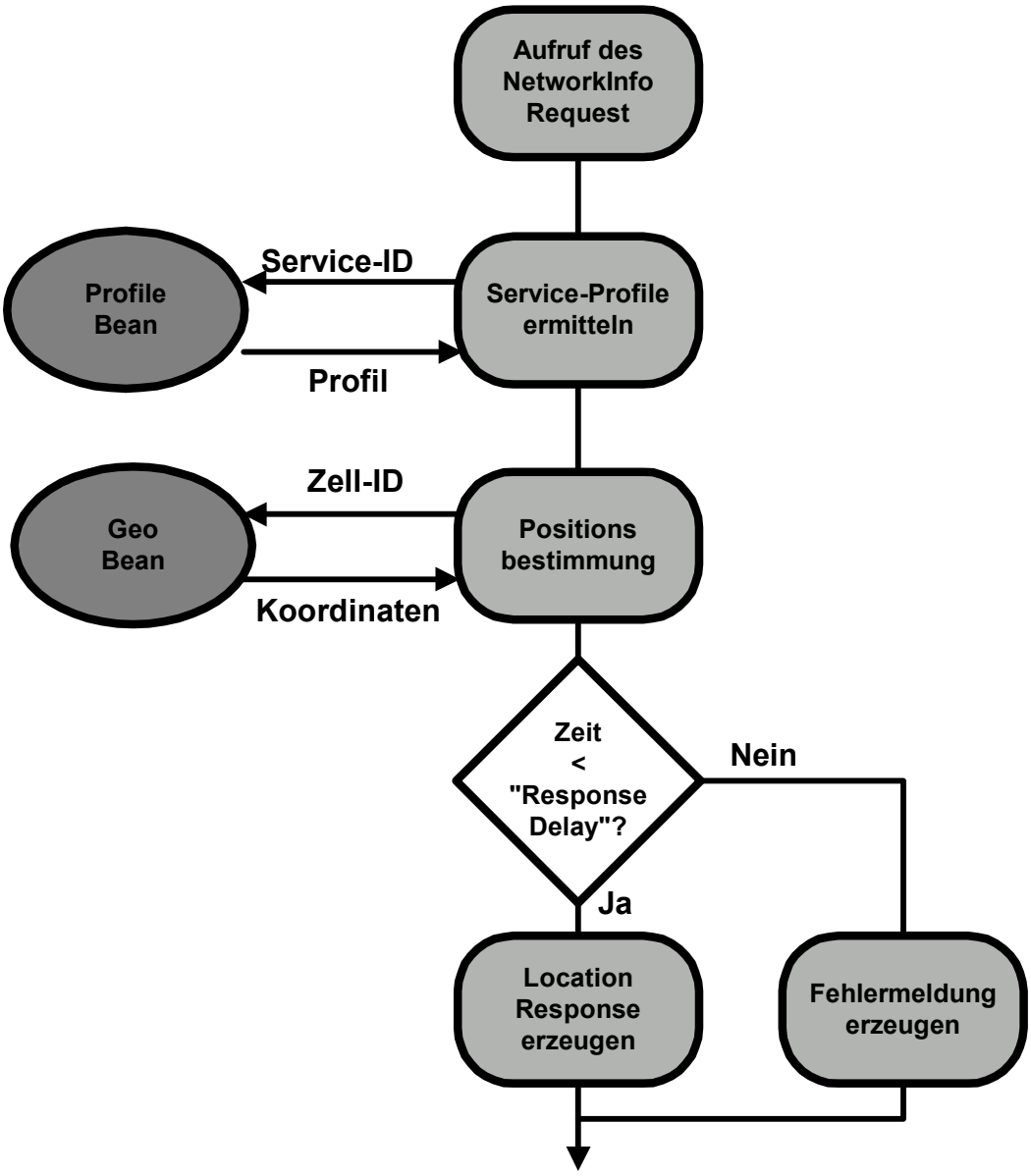

Abbildung 34 - Aktivitätsdiagramm eines NetworkInfo Requests

Analog zum Location Request wird zunächst das Service Profile anhand der Service-ID bestimmt. Hierzu wird auch im NetworkInfo Request auf die „Profile“-EntityBean zugegriffen, über die sämtliche Profilinformationen erfragt werden können. Nachdem auf diese Weise das Positionsformat ermittelt wurde, erfolgt zum Zweck der Koordinatenbestimmung, der Zugriff auf die „GeoData“-EntityBean. Schließlich wird die Laufzeit der Prozedur mit dem Wert verglichen, der durch den Parameter „RESPONSE_DELAY“ vorgegebenen wurde. Der

VASP erhält als Antwort auf seine Anfrage die „NetworkInfo Response“-Nachricht.

Beispiel: NetworkInfo Response

```
<MESSAGE_TYPE=15><REQUEST_ID=1><SESSION_HANDLE=2854>
<GEO_REF_SYSTEM=1><POSITION_FORMAT=0><NETWORKINFO_ID_TYPE=0>
<REQUEST_TIME=20021010><RESPONSE_DELAY=10000>
<GMT_DIFFERENCE=2><MCC=49><MNC=12><LAC=221><CELLID=1001>
<NETWORKINFO=<POSITION_FORMAT=CIRCLE;CENTER=555555,133333;RADI
US=646>>
```

Tabelle 11 - NetworkInfo Response

6.3.4.7 CIA

Das Enabling Produkt „CIA“ wird in der Simulation durch die CIA Entität[48] abgebildet. Zweck der Customer Identity Applikation ist die Ermittlung einer MSISDN oder Subscriber-Id auf Basis einer IP-Adresse. Zur Bereitstellung dieser Information wird der „Identity Request“[49]-Dienst implementiert.

Beispiel: Identity Request

```
<MESSAGE_TYPE=100><REQUEST_ID=1><SESSION_HANDLE=2857>
<SERVICE_ID=1><CLIENT_IP_ADDRESS=194.64.38.2>
```

Tabelle 12 - Identity Request

Beim Identity Request erteilt der Parameter Service-Id Auskunft über den Rückgabewert des Dienstes und über die Herkunft des Clients. Als Rückgabewert kann die MSISDN oder Subscriber-Id definiert werden. Beim Client unterscheidet man zwischen einem externen VASP und einer internen Applikation. Letztere

[48] siehe Klassendiagramm in Abbildung 49 in Anhang D
[49] siehe Sequenzdiagramm in Abbildung 66 in Anhang D

spielt für die Simulation jedoch eine untergeordnete Rolle, da als Client grundsätzlich der VASP auszumachen ist.

Die CIA-Entität ist als zustandslose Session Bean realisiert und enthält in dieser Eigenschaft die Geschäftslogik des Enabling Produktes. Die persistenten Daten, IP-Adresse und MSISDN, werden hingegen von der „CallData“[50] Entity Bean gekapselt. Konnte die IP-Adresse ermittelt werden, so wird die „Identity-Response“-Nachricht an den VASP geschickt.

Beispiel: Identity Response

```
<MESSAGE_TYPE=110><REQUEST_ID=1><SESSION_HANDLE=2857>
<STATUS=0><CLIENT_IP_ADDRESS=194.64.38.2>
<KIND_OF_IDENTITY=10><IDENTITY=01715456399>
```

Tabelle 13 - Identity Response

[50] siehe Klassendiagramm in Abbildung 46 in Anhang D

6.4 Implementierungshinweise

6.4.1 Programmierumgebung

6.4.1.1 Ant

Ant ist ein auf Java basierendes Build-Tool, das von Apache entwickelt wurde. Ein Build-Tool kontrolliert das Erstellen von ausführbaren Dateien aus dem Quellcode der Datei. Ein Endbenutzer kann so lauffähige Dateien aus dem Quellcode generieren, ohne irgendwelche Kenntnisse über das Programmieren zu besitzen. Für Entwickler kann ein Build-Tool hilfreich sein, da das ständige Kompilieren und Verpacken der Binärdateien stark vereinfacht wird.

Das bekannteste Build-Tool ist *Make*. Make arbeitet mit verschiedenen Shell-Skripten und weist daher eine große Fehleranfälligkeit sowie einen hohen Pflegeaufwand auf. Eine der Stärke von Apache Ant ist hingegen die XML-basierte Konfiguration, welche im Vergleich zu einer *Makefile* Konfiguration viel einfacher les- und wartbar ist. In einer zentralen XML-Datei lassen sich einzelne „Tasks“ und Abhängigkeiten zwischen diesen definiert. Ein weiterer Vorteil von Ant besteht in der Plattformunabhängigkeit des Tools.

Aus dem Build-Prozess mit Apache Ant kann eine Vielzahl von Aufgaben angestoßen werden. Dazu gehören beispielsweise

- das Aktualisieren der Quelldateien zum Repository,
- die Kompilation der Quelldateien,
- das Hochfahren bzw. Herunterfahren des Applikationsservers,
- der Test von Sourcen mit dem aktiven Applikationsserver,
- die Erstellung eines Testprotokolls,
- sowie das Versenden der Testprotokolle an eine definierte Benutzergruppe.

Von großen Nutzen ist auch die Integrationsfähigkeit von Apache Ant in verschiedene IDEs wie z.B. Visual Age, Forte 4 Java und JBuilder. Hierdurch kann der Build-Prozess von einer graphischen Oberfläche gesteuert werden.

Das folgende Listing zeigt einen Auszug aus einer Ant Build Datei.

Beispiel: ANT Build Datei

```
<project name="MyProject" default="dist" basedir=".">

  <!-- set global properties for this build -->
  <property name="src" value="."/>
  <property name="build" value="build"/>
  <property name="dist"  value="dist"/>

  <target name="init">
    <!-- Create the time stamp -->
    <tstamp/>
    <!-- Create the build directory structure used by compile -->
    <mkdir dir="${build}"/>
  </target>

  <target name="compile" depends="init">
    <!-- Compile the java code from ${src} into ${build} -->
    <javac srcdir="${src}" destdir="${build}"/>
  </target>

  <target name="dist" depends="compile">
    <!-- Create the distribution directory -->
    <mkdir dir="${dist}/lib"/>

    <!-- Put everything in ${build} into the MyProject-${DSTAMP}.jar file -->
    <jar jarfile="${dist}/lib/MyProject-${DSTAMP}.jar" basedir="${build}"/>
  </target>

  <target name="clean">
    <!-- Delete the ${build} and ${dist} directory trees -->
    <delete dir="${build}"/>
    <delete dir="${dist}"/>
  </target>
</project>
```

Tabelle 14 - Ant Datei

Über das „property“-Tag lassen sich Variablen definieren, die von verschiedenen Targets benötigt werden. Ein Target repräsentiert eine auszuführende Aufgabe innerhalb des Build-Prozesses. Zwischen den Targets lassen sich über das Attribut „depends“ Abhängigkeiten definieren. So muss beispielsweise das Target „compile“ ausgeführt werden, bevor das Verteilen („dist“) der Klassen erfolgt.

6.4.1.2 XDoclet

XDoclet ist ein Tool, welches die „Javadoc Doclet Engine“ erweitert und eine attribut-orientierte Programmierung mit Java ermöglicht. Quellcode-Dateien werden um Tags und Tag-Attribute ergänzt, die einem Objekt zusätzliche Eigenschaften zuweisen. So können beispielsweise für eine Session Bean Informationen bezüglich der Zugriffsart (remote oder local), der Zustandshaltung (stateless oder stateful) oder des JNDI-Namens spezifiziert werden. Das folgende Listing zeigt die Spezifikation von Tags für die CRISP Session Bean.

Beispiel: CRISP Session Bean

```
import com.tmobile.ns.util.MobileDetails;
import com.tmobile.ns.util.ParameterEvaluator;

/**
 * @ejb:bean type="Stateless" name="Crisp" jndi-name="ejb/Crisp" view-
type="remote"
 * @ejb:ejb-ref ejb-name="LBS"
 * @ejb:ejb-ref ejb-name="CIA"
 * @ejb:ejb-ref ejb-name="MsgProducer"
 * @ejb:ejb-ref ejb-name="UserData"
 * @ejb:ejb-ref ejb-name="MobileStation"
 * @ejb:transaction type="Required"
 * @ejb:transaction-type type="Container"
 *
 * @jboss:ejb-ref-jndi ref-name="LBS" jndi-name="ejb/LBS"
 * @jboss:ejb-ref-jndi ref-name="CIA" jndi-name="ejb/CIA"
 * @jboss:ejb-ref-jndi ref-name="MsgProducer" jndi-name="ejb/MsgProducer"
 * @jboss:ejb-ref-jndi ref-name="UserData" jndi-name="ejb/UserData"
 * @jboss:ejb-ref-jndi ref-name="MobileStation" jndi-name="ejb/MobileStation"
 *
 */
public class CrispBean implements SessionBean {
    private static Hashtable hSessionHandle = new Hashtable();
    private SessionContext ctx;
    private LBSLocalHome lbsHome = null;
    private CIALocalHome ciaHome = null;
    ...
```

Tabelle 15 - XDoclet Attribute im EJB-Code

XDoclet ist Teil des in Kapitel 6.4.1.1 beschriebenen Ant-Prozesses. Durch die Definition von XDoclet spezifischen Tasks innerhalb der Ant Build-Datei, werden verschiedene XDoclet Prozesse angestoßen. Basierend auf diesen Tasks sowie den Tags und Tag-Attributen, die einem Objekt zugewiesen werden, lassen sich unterschiedliche Objekten generieren. Zu diesen Objekte zählen u.a.

- Remote- und Local-Interfaces für Enterprise Beans,
- Data Access Objects,
- Util-Klassen für EnterpriseBeans,

- und Deployment Deskriptoren.

Der Vorteil des XDoclet-Tools wird deutlich wenn man bedenkt, das ein Entwickler nur eine einzige Datei anlegen und pflegen muss, aus der sich dann eine Vielzahl anderer Objekte ableiten lassen. Ein weiterer Mehrwert ergibt sich durch den Einsatz von herstellerspezifischen Tasks in der Ant Build-Datei. Diese ermöglichen es beispielsweise Deployment Deskriptoren für unterschiedliche Applikationsserver zu erzeugen. Hierdurch wird die Portierbarkeit der Anwendung gewährleistet.

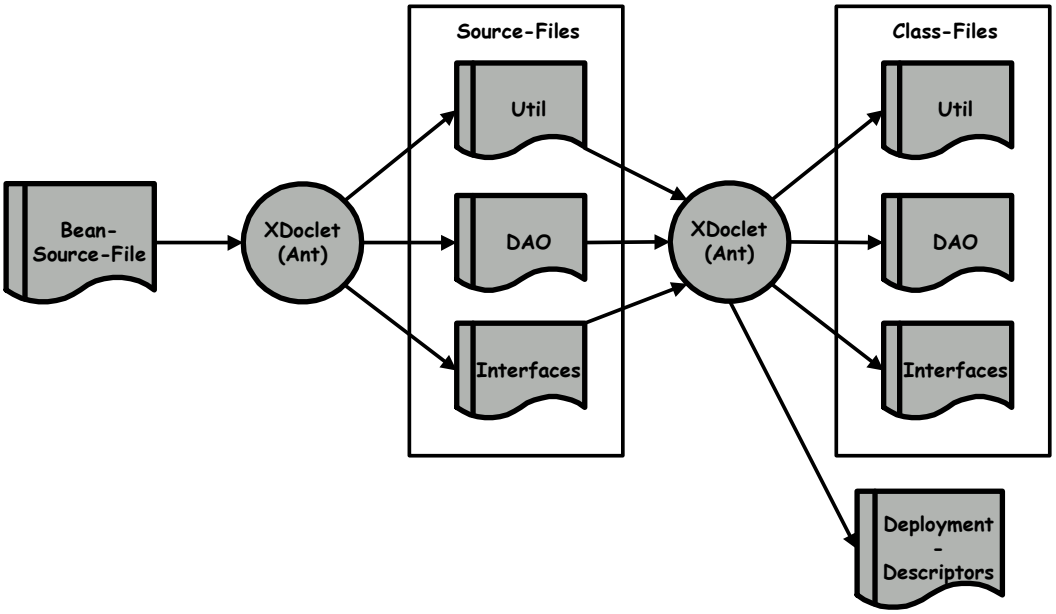

Abbildung 35 - Der XDoclet-Prozess

6.4.2 Systemumgebung

6.4.2.1 Java 2 Standard Edition (J2SE)

Die Java 2 Standard Edition besteht im wesentlichen aus zwei Komponenten: der Java Virtual Maschine (JVM) und dem Java Application Programming Interface (API). Die Java Virtual Maschine bildet die Basis zur Ausführung von Java

Programmen. Sie interpretiert den vom Compiler erzeugten Bytecode und übersetzt diesen in den Maschinencode des jeweiligen Betriebssystems. Die Java API stellt hingegen eine Abstraktion verschiedener Dienste vom Betriebssystem dar. So werden beispielsweise in der Bibliothek „java.awt.*“ Klassen zur Programmierung von grafischen Oberflächen bereitgestellt.

Für die Realisierung des Netzsimulators wurde die Java 2 Standard Edition in der Version 1.4.0 eingesetzt. Sie stellt gegenüber der Vorgängerversion einige neue Leistungsmerkmale zur Verfügung. Dazu gehören u.a. Erweiterungen der Java Foundation Classes (JFC) sowie neue Funktionalitäten im Bereich der XML-Verarbeitung. Innerhalb des Steuerungsinterfaces konnte daher eine verbesserte „Drag and Drop“-Funktionalität der JFC genutzt werden. Bei der Verarbeitung von XML-Dokumenten musste nicht auf externe Tools zurückgegriffen werden, da sich bereits alle XML-Standards (SAX 1.0 und 2.0, DOM 1.0 und 2.0, XSLT) in der API wiederfanden.

6.4.2.2 Tomcat 4.1

Zur Bereitstellung von Web-Inhalten und zum Austausch von Informationen über das HTTP-Protokoll existiert in der Web-Tier ein Webserver. Durch das Server Application Programming Interface (SAPI) stellt der Webserver erweiterte Funktionalitäten zur Verfügung. Auf diese Weise können dynamische Informationen in statische HTML-Seiten integriert werden.

Als Webserver Produkt wird der Apache Tomcat Webserver in der Version 4.1 eingesetzt. Dabei handelt es sich um einen Open Source Container für Java basierte Web-Anwendungen, der Servlets und JSP-Applikationen ausführt. Tomcat kann leicht für verschiedene Einsatz-Umgebungen konfiguriert werden. Neben der Verwendung als alleinstehenden Webserver ist auch die Integration als Servlet/JSP Umgebung in einen bestehenden Webserver oder Applikationsserver möglich. In der hier beschriebenen Systemumgebung ist Tomcat in den Applikationsserver „JBoss“, der im folgenden Kapitel erläutert wird, integriert.

Die eingesetzte Version 4.1 verfügt im Gegensatz zur Vorgängerversion über eine neue Architektur, die vollständig komponentenbasiert ist. Zudem löst der Catalina Servlet Container das älter JServ-Modul ab. Ziel der neuen Architektur ist eine bessere Verständlichkeit der Server-Operationen sowie eine größere Flexibilität bei der Konfiguration.

6.4.2.3 JBoss 3.0

Applikationsserver stellen ein Ausführungsumfeld für die Geschäftslogik zur Verfügung und verbergen die Komplexität der verteilten Umgebung. Ein Applikation Server entspricht einem Framework, das auf einer oder mehreren definierten Plattformen basiert und sämtliche systemorientierten Dienste bereitstellt, die zur Ausführung von Geschäftslogik erforderlich sind.

Der hier eingesetzte Applikationsserver „JBoss“ ist ein Open Source Produkt, das vollständig in Java implementiert ist. Schnittstellen und Architektur von JBoss entsprechen der Enterprise Java Bean Spezifikation. JBoss zeichnet sich vor allem durch den JMX-Server (Java Management Extension) aus. JMX, ein von Sun als offizielle J2EE-Komponente vorgestelltes Management-API, wird sich als Standard fürs Management verteilter Java-Applikationen immer mehr durchzusetzen, so dass auch die kommerziellen Anbieter wie BEA mit Weblogic nachziehen und JMX integrieren. Der Vorteil dieser Technologie ist ein modularer Aufbau mit klaren Schnittstellen, was ein einfaches Einbinden zusätzlicher Komponenten und Erweiterungen erlaubt.

Zur Zeit existieren JMX-Wrapper für zwei Webserver/Servlet-. Der Jetty-Webserver gehört zum Standart und ist in den JBoss-Applikation Server integriert. Alternativ dazu kann JBoss auch mit Tomcat, als externem Webserver, verwendet werden. Die folgende Abbildung zeigt das Zusammenspiel zwischen JBoss, Tomcat und der im nächstem Kapitel beschriebenen Datenbank Hypersonic.

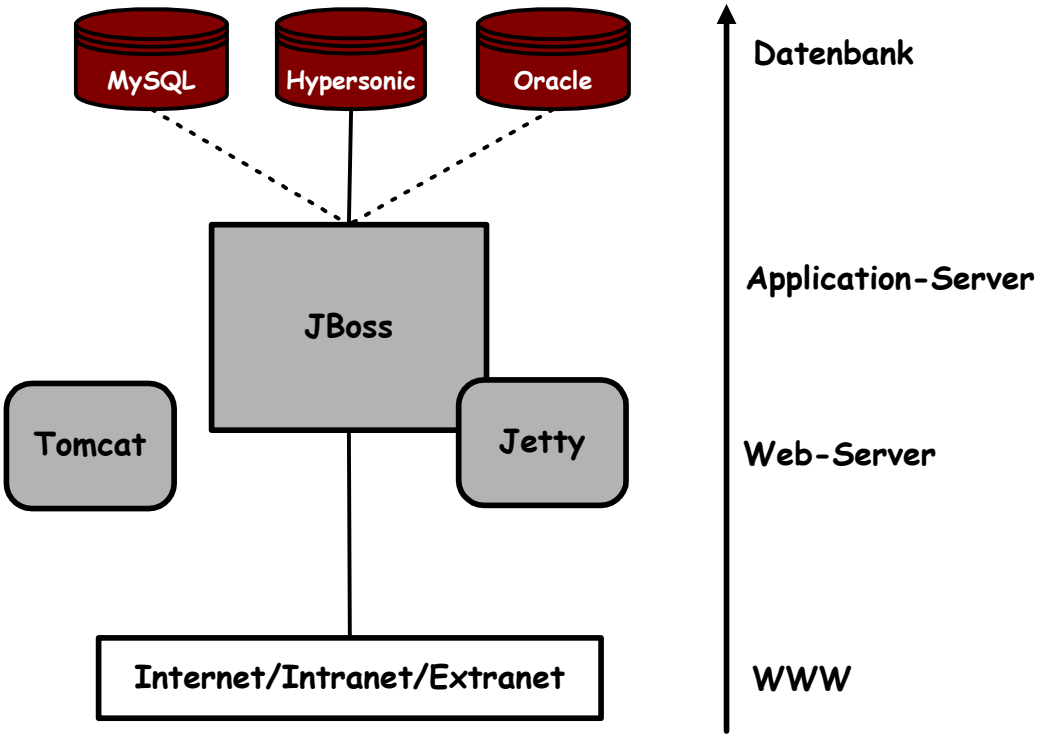

Abbildung 36 - Das JBoss-Umfeld

In der Version 3.0 unterstützt JBoss EJB 2.0 und Container Managed Persistence 2.0 (CMP). Weitere Neuheiten sind die Unterstützung von Clustering, JINI sowie Message Driven Beans.

6.4.2.4 Hypersonic

Das hier eingesetzte Datenbanksystem “Hypersonic” ist eine relationale Datenbank, die standardmäßig mit JBoss ausgeliefert wird. Die Datenbank ist vollständig in Java implementiert, wodurch auch innerhalb der Resource-Tier die Portierbarkeit des Simulators gewährleistet wird. Das Hypersonic Projekt, das ursprünglich von Thomas Müller gestartet wurde, ist mittlerweile eingestellt. Es existiert jedoch ein Nachfolgeprojekt mit den Namen „hsql Database Engine (hsqldb)“. Da der Netzwerksimulator auf der Basis der *Container Managed Persistence* (vergleiche Kapitel 5.1.5.1) entwickelt wurde, ist es ohne großen

Aufwand möglich auf andere relationale Datenbanksystem (z.B. Oracle oder MySQL) zu migrieren.

Kapitel 7

Ausblick

In diesem Kapitel sollen abschließend mögliche wirtschaftliche Entwicklungen sowie technische Erweiterungsmöglichkeiten des Netzsimulators dargestellt werden.

7.1 Wirtschaftliche Perspektive

Die Erwartungen der Betreiber in Bezug auf Non-Voice Dienste sind beträchtlich. Das diese nicht unbegründet sind, zeigen zahlreiche Marktstudien unterschiedlicher Analysten, die diesem Geschäftsfeld ein hohes Wachstum prophezeien. Damit die gesteckten Erwartungen aber auch umgesetzt werden, müssen noch eine ganze Reihe von Hürden genommen werden. So ist neben dem perfekten Zusammenspiel von Mobilfunkbetreibern und Ausrüstern, auch die Einbindung der Value Added Service Provider ein Schlüsselkriterium, um eine kritische Masse bzw. Marktpenetration zu erreichen.

Obwohl der Bedarf beim Kunden sicher da ist, muten jedoch viele der Dienste eher als "Spielerei" an. Hier gilt es die Spreu vom Weizen zu trennen und kreative Anwendungen zu entwickeln, die sich tatsächlich an den Bedürfnissen der Kunden orientieren. Dabei sollte man sich aber nicht auf einzelne Killerapplikationen fixieren, da dieser Weg für den Anbieter leicht in einer Sackgasse enden kann. Sinnvoller erscheint es die Möglichkeiten und Synergien unterschiedlicher Plattformen auszuschöpfen und dabei auch nicht die bereits gängigen Dienste, wie „Sprachkommunikation“ und „SMS“, außer Acht zu lassen. Diese Dienste werden auch in Zukunft einen beträchtlichen Umsatzanteil generieren und können darüber hinaus als Ausgangspunkt eines Migrationspfades (z.B. über EMS) zu neuen Multimediadiensten betrachtet werden.

Laut einer Studie von BBDO Consulting, stellt aus Verbrauchersicht die mobile Information (z.B. Börsenkurse oder Reiseinformationen) die attraktivste Anwendung dar. An zweiter Stelle stehen mobile Transaktionen wie Online-Banking oder Online-Shopping. Mobiles Entertainment gehört für die Verbraucher derzeit noch nicht zu den begehrtesten Anwendungen. Abbildung 37 illustriert diese Nutzungswahrscheinlichkeiten.

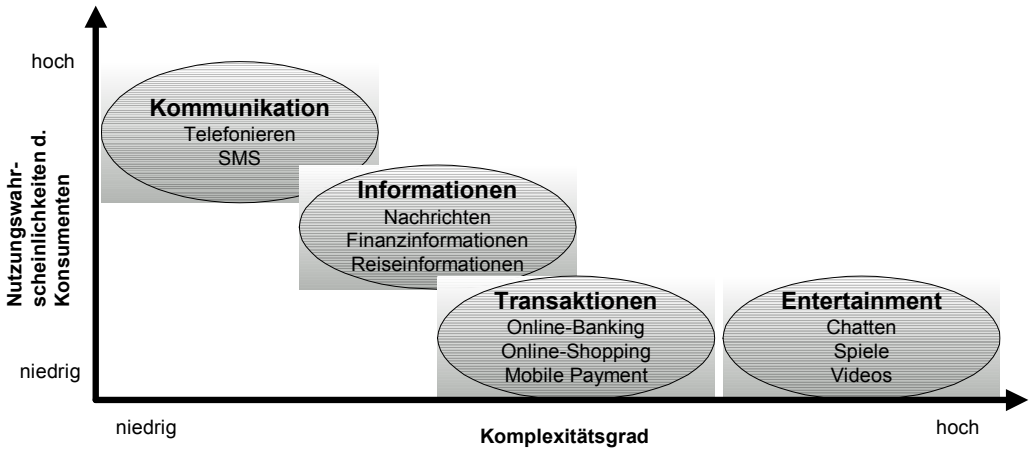

Abbildung 37 - Nutzungswahrscheinlichkeit der Konsumenten

7.2 Technische Perspektive

Non-Voice-Anwendungen bzw. mobile IT-Anwendungen sind meist verteilte Anwendungen, die aus verschiedenen in sich abgeschlossenen Komponenten bestehen und oft auf verschiedenen Plattformen oder an unterschiedlichen Orten betrieben werden. Ein VASP der standortbezogene Informationen bereitstellt, muss beispielsweise auf die externen Daten eines LBS-Servers zugreifen. Da die meisten VASPs mit verschiedenen Netzbetreiber kooperieren, ergibt sich an dieser Stelle das Problem, das keine Standardschnittstellen existieren. Jede Schnittstelle muss in einer vorher definierten Sprache implementiert werden. Die Aufrufe der einzelnen Methoden sind sprach- und/oder protokollabhängig.

Ein Schlagwort das seit einiger Zeit stets im Zusammenhang mit verteilten Anwendungen zu hören ist, sind Web Services. Durch Web Services soll eine sprach- und protokollunabhängige Möglichkeit geschaffen werden, Informationen auszutauschen und in bestehende Anwendungen zu integrieren. Dabei werden sprachabhängige Methodenaufrufe in sprachunabhängige XML-Dokumente transformiert und am Ziel wieder sprachabhängige Konstrukte gewandelt.
Ein Web Service kann eine Geschäftseinheit, eine Anwendung oder System-Funktionalität, die über das WWW erreichbar ist, repräsentieren.

Besonders für mobile Internet-Anwendungen, bei denen Informationen jederzeit, an jedem Ort und auf jedem Geräte verfügbar sein müssen, stellt die protokoll- und sprachenunabhängige, systemübergreifende Kommunikation mit Web Services ein unverzichtbares Rüstzeug dar.

Abbildung 38 zeigt die Technologien, die bei der Verwendung von WebServices zum Einsatz kommen:

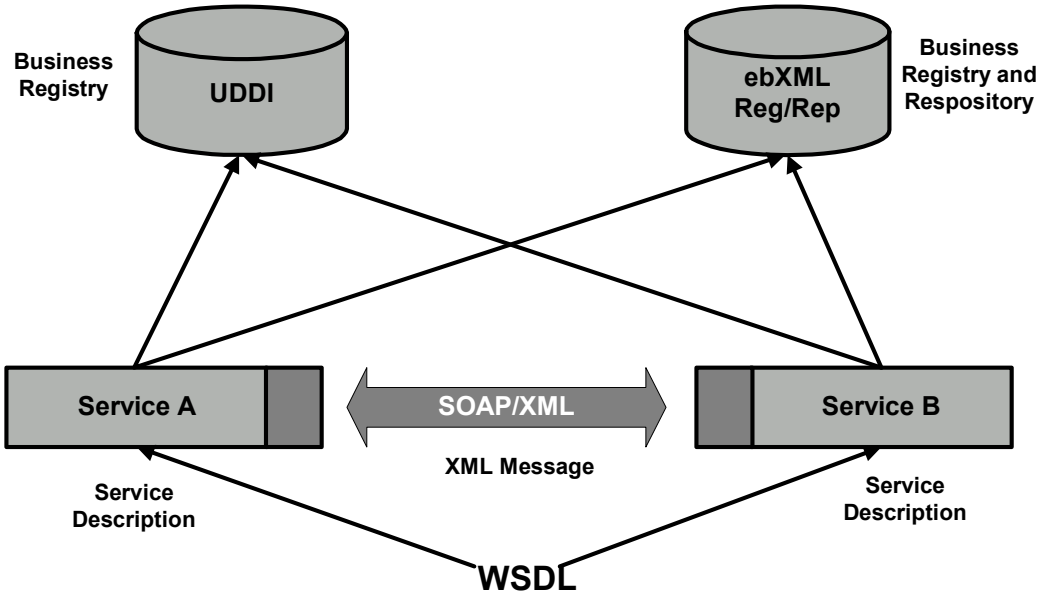

Abbildung 38 - Technologien im WebService-Umfeld

- **UDDI**: Universal Description Discovery and Interoperability ist eine Initiative um ein universelles webbasiertes Geschäftsverzeichnis zu erstellen.

- **WSDL**: Web Services Description Langauge ist eine Beschreibungssprache, die dafür verwendet wird, um die Art des Aufrufs und die Ausführung eines Web Services zu definieren.

- **SOAP**: Das Simple Object Access Protokoll ermöglicht das Senden von XML-Konstrukten über das HTTP-Protokoll. Aus sprachabhängigen Objekten werden dabei sprachunabhängige XML-Konstrukte.

- **ebXML** ist ein komplettes Business-To-Business-Framework, das Business Kollaboration durch Teilen von webbasierten Business-Services ermöglicht.

Anhang A - Literaturverzeichnis

[1] **BBDO Consulting** "mCommerce mit UMTS"<http://www.bbdo.de/bbdo-media/umts.pdf>(07.04.2002)

[2] **Biala, Jacek :** Mobilfunk und Intelligente Netze - Grundlagen und Realisierung mobiler Kommunikation, 1995

[3] **Blank, Hans-Joachim; Janik, Dr. Hubert:** Telekommunikation für Profis, 1995
Gabhart, Kyle : Professional EJB, 2002.

[4] **Institut für Zukunftsforschung und Technologiebewertung:** Entwicklung und zukünftige Bedeutung mobiler Multimediadienste <http://www.izt.de/publikationen/#zukunftsstudien>(07.04.2002)

[5] **Kluschke, Michael; Wölfel Ludgar:** Web Services als E-Business-Evolution, in: Magazin für professionelle Informationstechnik (IX) 11/2001, S. 149-154

[6] **Monson-Haefel, Richard :** Enterprise JavaBeans, 2001

[7] **Paine, Stephen; Burnett, Steve:** Kryptographie, 2001

[8] **Roßbach,Peter; Schreiber,Hendrik :** Java Server und Servlets, 1999

[9] **Saleck, Krister; Turau, Volker; Schmidt, Marc:** Java Server Pages und J2EE, 2001

[10] **Seiler, Bernd:** Taschenbuch der Telekompraxis, 2002

[11] **Simers, Sybille:** MVC- meets XML, in: Java Magazin 03/2002, S. 24-30

[12] **Steinbach,Christine:** Mehrwertdienste, 1993

[13] **Walke, Bernhard:** Mobilfunknezte und ihre Protokolle, 2001

[14] **Wizki, Axel:** Location Based Services, in: funkschau - Professionelle Telekommunikation 05/2002

Anhang B - Inhalt der CD-ROM

/doc

Dieses Verzeichnis enthält die vorliegende Diplomarbeit im PDF-Format. Darüber hinaus enthält es eine Installationsversion des Programms Acrobat Readers.

/dev

In diesem Verzeichnis befinden sich neben den entwickelten Quellcode-Dateien, die zur Weiterentwicklung benötigten werden, die Programme „XDoclet“ und „Ant“.

/bin

Dieses Verzeichnis enthält die lauffähigen Komponenten des Netzwerksimulators sowie weitere Programme und Tools, die zur Ausführung der Simulation erforderlich sind.

Anhang C - Installationsanleitungen

Zur Installation des Netzsimulators sind die folgenden Anweisungen in der angegebenen Reihenfolge auszuführen.

1. **Java 2 SDK 1.4 Installation**

   Auf der CD befinden sich im Verzeichnis "/bin/j2sdk" verschiedene JDK Distributionen. Installieren sie das für Ihr Betriebssystem entsprechende JDK auf ihrer Festplatte. Hierzu muss die jeweilige Datei lediglich extrahiert werden. Anschließend ist die Umgebungsvariable "JAVA_HOME" zu setzen.

   Beispiel:

   Linux: *export JAVA_HOME=/usr/java/j2sdk1.4/*

   Windows: Systemsteuerung → System → Erweitert → Umgebungsvariabeln → Systemvariabeln → Neu → „*JAVA_HOME = c:\java\j2sdk1.4*“

2. **JBoss 3.0 Installation**

   Im Verzeichnis "/bin/jboss" befindet sich der JBOSS Application Server. Dieser muss auf die Festplatte kopiert und extrahiert werden. Analog zur Variabeln „*JAVA_HOME*“ ist hier auch die Umgebungsvariable "*JBOSS_HOME*" zu setzen. Der Application Server kann anschließend über die Skripte „*run.bat (Windows)*“ bzw. „*run.sh (Unix)*“ im "*bin*"-Verzeichnis gestartet werden.

3. **Installation der VASP Applikation**

   Wechseln Sie in das Verzeichnis „/bin/vasp“ und kopieren Sie die Datei

„vasp.ear“ in das Deploy-Verzeichnis „*%JBOSS_HOME%/server/default/deploy“* des Applikationservers. Anschließend kann die VASP-Applikation über die entsprechende URL des Webservers gestartet werden.

Beispiel:

http://localhost:8080/vasp/servlet/VaspServlet

4. **Installation der Wirknetz-Simulation**

Gehen Sie hierzu analog zur Installation der VASP-Applikation vor. Wechseln Sie in das Verzeichnis „/bin/wirknetz“ und kopieren Sie die Datei „netSimulator.ear“ in das Deploy-Verzeichnis von JBoss.

5. **Installation des Steuerungsinterfaces**

Die Dateien im Verzeichnis "/bin/client" müssen auf die Festplatte kopiert werden. Wichtig ist hierbei das sich das Skript und JAR-File im selben Verzeichnis befinden. Das Steuerungsinterface kann dann unter Angabe des Users „*java“* sowie des Passwortes „*java4me“* über das jeweilige Skript gestartet werden.

Beispiel:

./runClient java java4me

Über das Skript können weitere User angelegt werden. Wird das Skript ohne Parameter aufgerufen, erscheinen die möglichen Aufrufoptionen.

6. **Installation des VASP-Zertifikats**

Das VASP-Zertifikat der Vasp Applikation muss im Server Truststore importiert werden. Der Server verwendet i. d. Regel die folgende Datei als Truststore:

Windows: *%JAVA_HOME%\jre\lib\security\cacerts*
Unix: *$JAVA_HOME/jre/lib/security/cacerts*

Das Zertifikat des Vasp („vasp.cert“) befindet sich auf der CD im Verzeichnis "/bin/vasp". Es kann durch die folgende Anweisung im Server Truststore importiert werden:

Windows: *%JAVA_HOME%\bin\keytool -import -file vasp.cert -alias vasp -keystore %JAVA_HOME%\jre\lib\security\cacerts*

Unix: *$JAVA_HOME/bin/keytool -import -file vasp.cert -alias vasp -keystore $JAVA_HOME/jre/lib/security/cacerts*

Zertifikate anderer Clients können analog zum Vasp-Beispiel importiert werden.

7. **Installation der Server Keystore-Datei**

Als nächstes muss die Server Keystore-Datei in JBoss abgelegt werden. Diese Datei heißt „wirknetz.keystore“ und befindet sich auf der CD-ROM im Verzeichnis „/bin/wirknetz“. Kopieren Sie die Datei in das Verzeichnis „%JBOSS_HOME%/bin“.

Die folgende Abbildung verdeutlicht die Lokation, der unter Punkt 6 und 7 beschriebenen Zertifikate bzw. Keystore-Dateien und zeigt wie die

Informationen zwischen VASP und CRISP ausgetauscht werden.

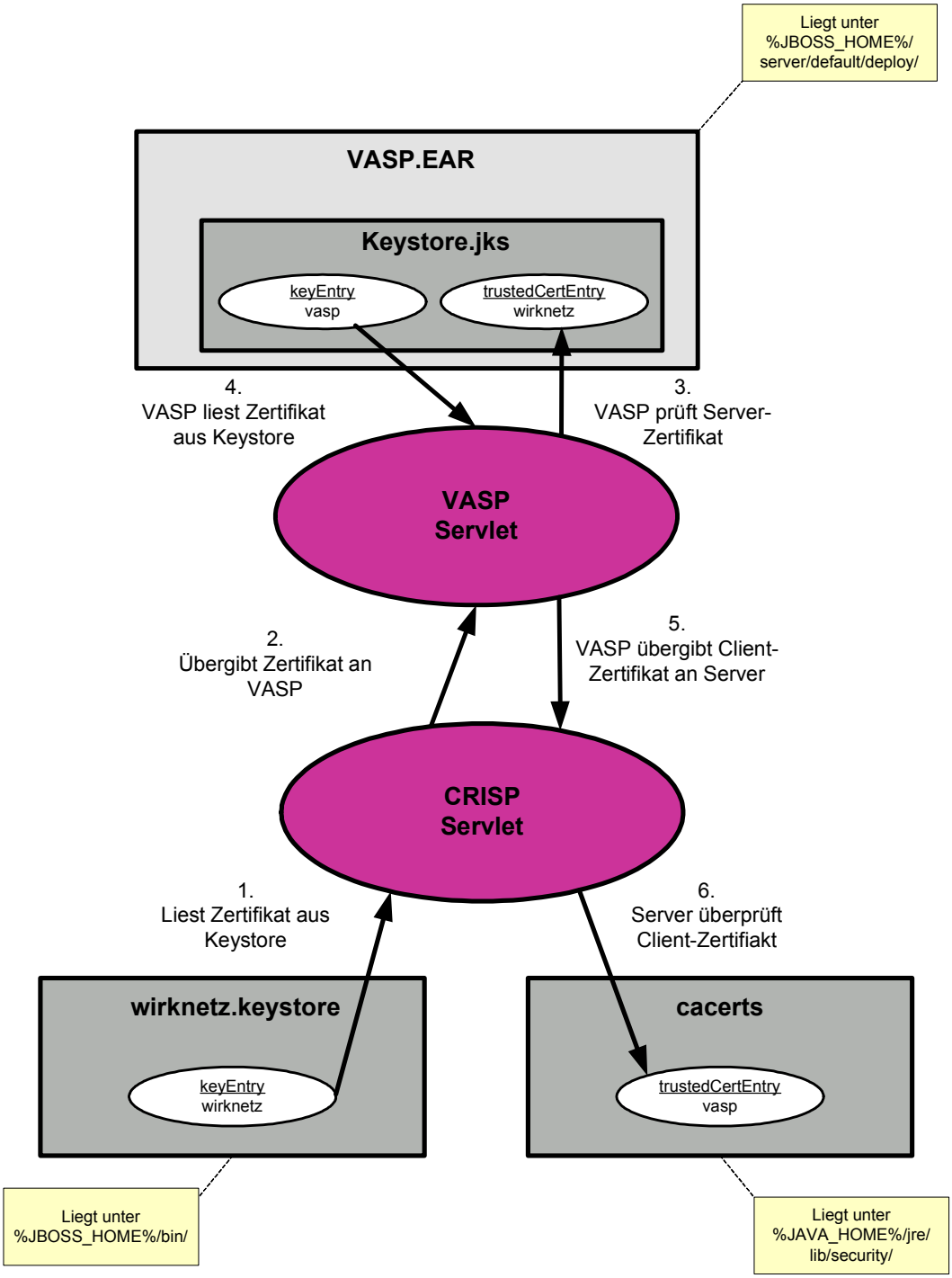

Abbildung 39 - Austausch der Zertfikatsdaten

8. Konfiguration von JBoss

Falls Sie JBoss gestartet haben, beenden Sie den Prozeß durch den Aufruf des Skriptes „shutdown.bat“ bzw „shutdown.sh“. Um SSL zu aktivieren, wechseln

Sie nun in das Verzeichnis „%JBOSS_HOME%/server/default/deploy“ und öffnen Sie die Datei „tomcat4-service.xml“. Fügen Sie den folgenden Eintrag zwischen das „Service“-Tag in der Datei.

JBoss SSL-Konfiguration

```
<!-- Define an SSL HTTP/1.1 Connector on port 8443 -->

   <Connector className=
              "org.apache.catalina.connector.http.HttpConnector"
               port="8443" minProcessors="5" maxProcessors="75"
               enableLookups="true"
               acceptCount="10" debug="0" scheme="https" secure="true">
               <Factory className=
                  "org.apache.catalina.net.SSLServerSocketFactory"
                  clientAuth="true"
                  protocol="TLS"
                  keystoreFile="../bin/wirknetz.keystore"
                  keystorePass="changeit"/>
   </Connector>
```

Starten Sie anschließend JBoss neu. Eine Beispieldatei finden Sie auf der CD-ROM im Verzeichnis „/bin/jboss“.

Die weiteren Anweisungen sind zum Einrichten der Entwicklungsumgebung von Bedeutung.

❖ **XDoclet Installation**

Die XDoclet Installationsdatei befindet sich im Verzeichnis „/dev/xdoclet“. Kopieren Sie die Datei auf die Festplatte und extrahieren Sie diese in ein beliebiges Verzeichnis. Setzen Sie anschließend die Umgebungsvariable „XDOCLET_HOME“ auf den Wert des Installationsverzeichnisses. Weitere Informationen zu XDOCLET finden Sie über die Online-Hilfe im „*doc*“-Verzeichnis.

❖ **ANT Installation**

Gehen Sie zur Installation von ANT genauso vor wie bei der XDOCLET - Installation. Kopieren Sie aus dem Verzeichnis „/dev/ant“ die Datei „ant.zip“ auf Ihre Festplatte und extrahieren Sie diese. Setzen Sie anschließend den Wert der Umgebungsvariabeln „ANT_HOME“ auf das entsprechende Verzeichnis.

❖ **Installation der Entwicklungsumgebung**

Sämtliche für die Entwicklung relevanten Dateien befinden sich im Verzeichnis „/dev/src“. Kopieren Sie dieses Verzeichnis auf Ihre Festplatte.

Zum Verständnis der im Folgenden beschriebenen Verzeichnisstruktur und Entwicklungskomponenten, ist es notwendig den Aufbau einer J2EE-Applikation zu kennen. Abbildung 39 zeigt aus welchen Elementen sich eine solche Anwendung zusammensetzt.

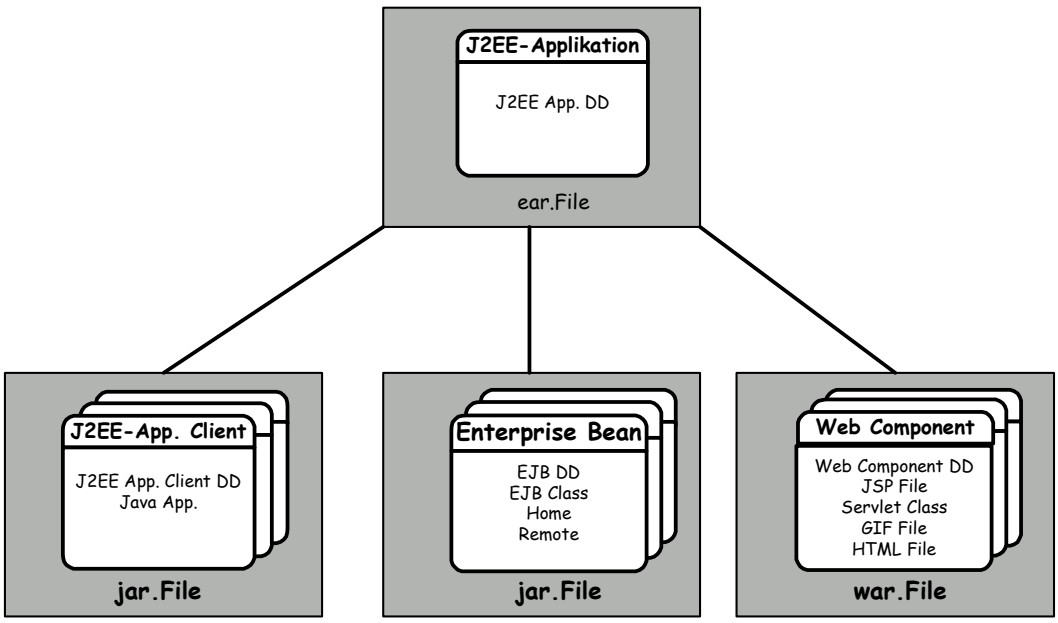

Abbildung 40 - Aufbau einer EAR-Datei

Wie der Grafik zu entnehmen ist, gibt es ein „*EAR*“-File, das verschiedene „*JAR*“-Dateien bzw. „*WAR*“-Dateien bündelt. Diese EAR-Datei enthält alle

Ressourcen, die zur Ausführung einer J2EE-Applikation notwendig sind. Zum „*deployen*“ einer J2EE-Anwendung muss daher nur die EAR-Datei in das jeweilige Verzeichnis des Applikationsservers gespielt werden. Die JAR- und WAR-Dateien entsprechen den in Kapitel 5 beschriebenen Schichten. Folgende Zuordnung kann gebildet werden:

- client.jar → Client-Tier
- ejb.jar → Business-Tier
- web.war → Web-Tier

Nicht alle Komponenten müssen in einem EAR-File enthalten sein. So kann die Datei beispielsweise nur aus den Elementen „ejb.jar“ und „web.war“ bestehen.

/vasp

Das „vasp“-Verzeichnis enthält die Entwicklungskomponenten für die VASP-Applikation. Die Struktur des Verzeichnisses ist in Abbildung 40 dargestellt und wird im Folgenden erläutert.

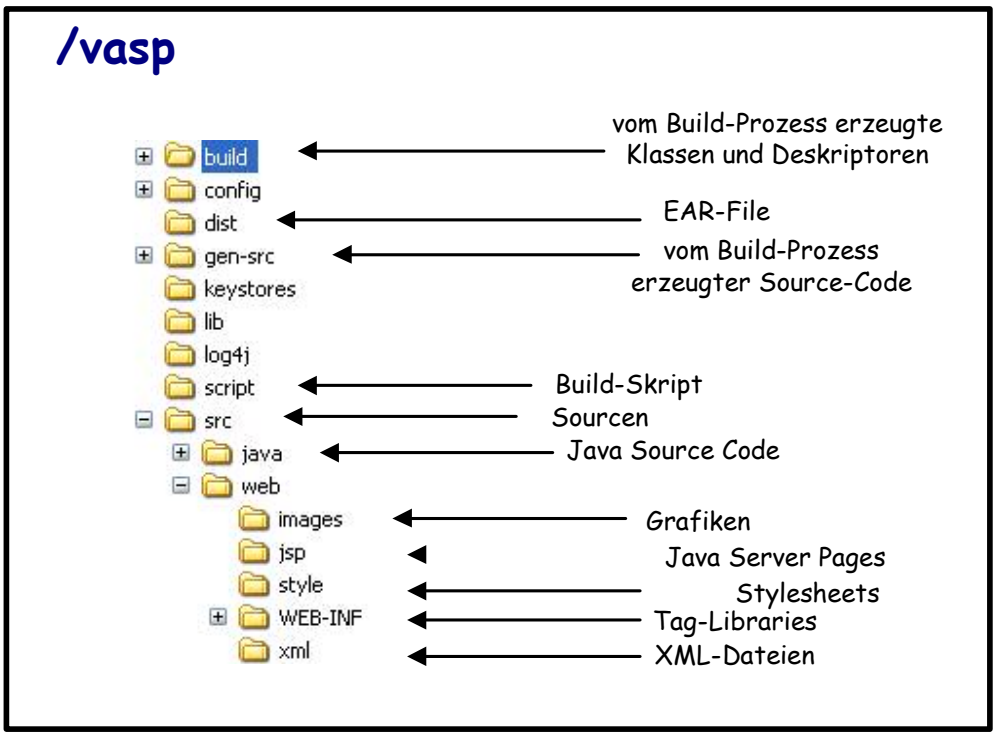

Abbildung 41 - Verzeichnis-Struktur der VASP-Applikation

Das Verzeichnis „*src*“ beinhaltet sämtliche Quelldateien. Neben dem Java-Code finden sich hier auch andere Ressourcen wie Bilder und Tag-Libraries wieder. Eine Änderungen an einer der Quelldateien macht ein Neu-Kompilieren erforderlich. Dies geschieht durch das Ausführen des Skriptes „*build.bat*“ bzw. „*build.sh*“ im Verzeichnis „*script*“. Das Skript stößt den XDoclet-Prozeß an und erzeugt im ersten Schritt weitere Quellcode-Dateien (z.B. Interfaces), die im Verzeichnis „*gen-src*“ abgelegt werden. Im nächsten Schritt werden dann aus den ursprünglichen und den erzeugten Quellcode-Dateien die Klassen und Deployment-Deskriptoren generiert und im Verzeichnis „*build*“ abgelegt. Zuletzt werden alle Dateien, die zur Programmausführung erforderlich sind, in ein EAR-File gepackt. Diese Datei befindet sich im Verzeichnis „*dist*“.

/wirknetz

Das „wirknetz“-Verzeichnis enthält alle Elemente, die für die Wirknetz-Simulation und das Steuerungsinterface von Bedeutung sind. Abbildung 41 zeigt die Struktur des Verzeichnisses.

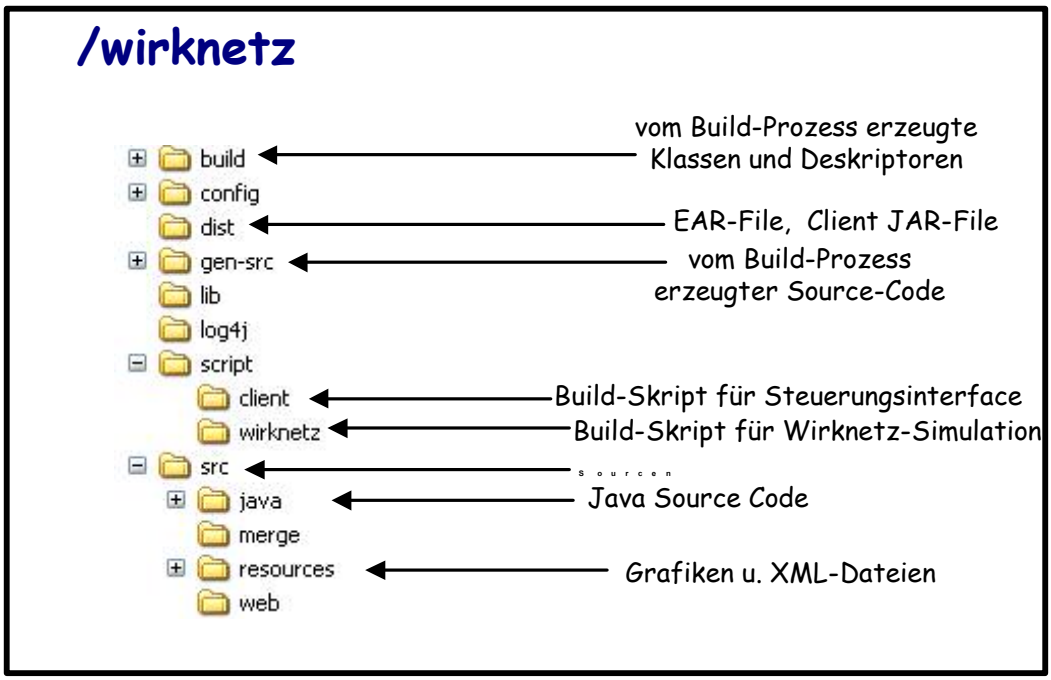

Abbildung 42 - Verzeichnis-Struktur der Wirknetz-Simulation

Die Verzeichnisstruktur entspricht im Wesentlichen der des „vasp"-Verzeichnisses. Bemerkenswert ist lediglich die Tatsache, dass es hier zwei Build-Skripte gibt. Im Verzeichnis „*client*" befindet sich das Build-Skript des Steuerungsinterfaces. Das Verzeichnis „*wirknetz*" enthält hingegen das Skript für die Wirknetz-Simulation. Folglich werden im Verzeichnis „*dist*" auch zwei Dateien erzeugt. Dies ist einerseits das EAR-File, welches alle Ressourcen der Wirknetz-Simulation bündelt. Andererseits existiert ein Client JAR-File, das sämtliche für das Steuerungsinterface relevanten Dateien enthält.

❖ **Deployment**

Wie bereits im vorigen Abschnitt beschrieben, muss bei einer Änderung der Quelldateien der XDoclet-Prozeß ausgeführt werden. Dies geschieht durch den Aufruf des jeweiligen Skriptes.

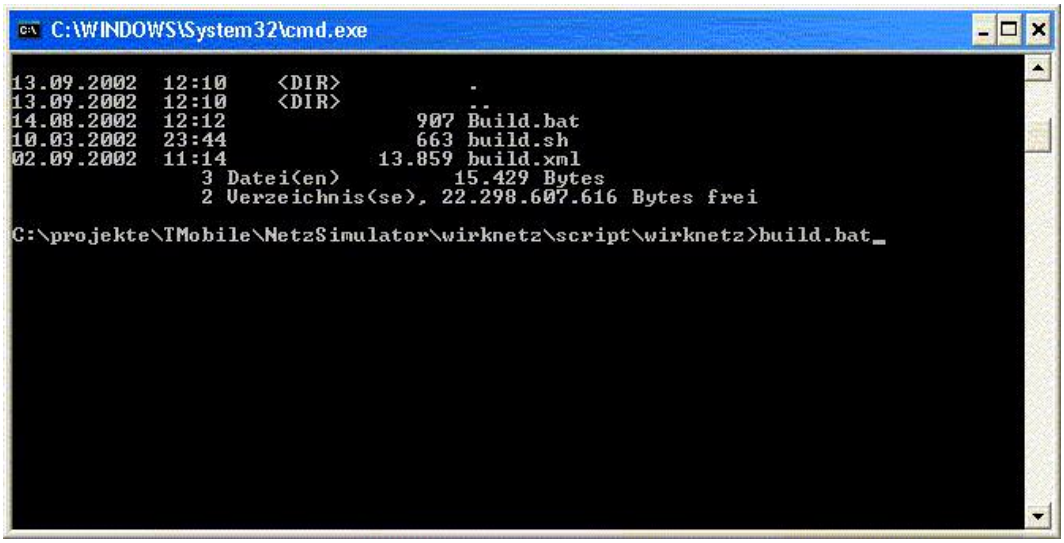

Abbildung 43 - Starten des XDoclet-Prozesses

Konnte der Prozess ohne Fehlermeldungen beendet werden, wird als Resultat ein EAR-File erzeugt und im Verzeichnis „*dist*" abgelegt. Diese Datei muss nun in das „*deploy*"-Verzeichnis des Applikationsservers kopiert werden.

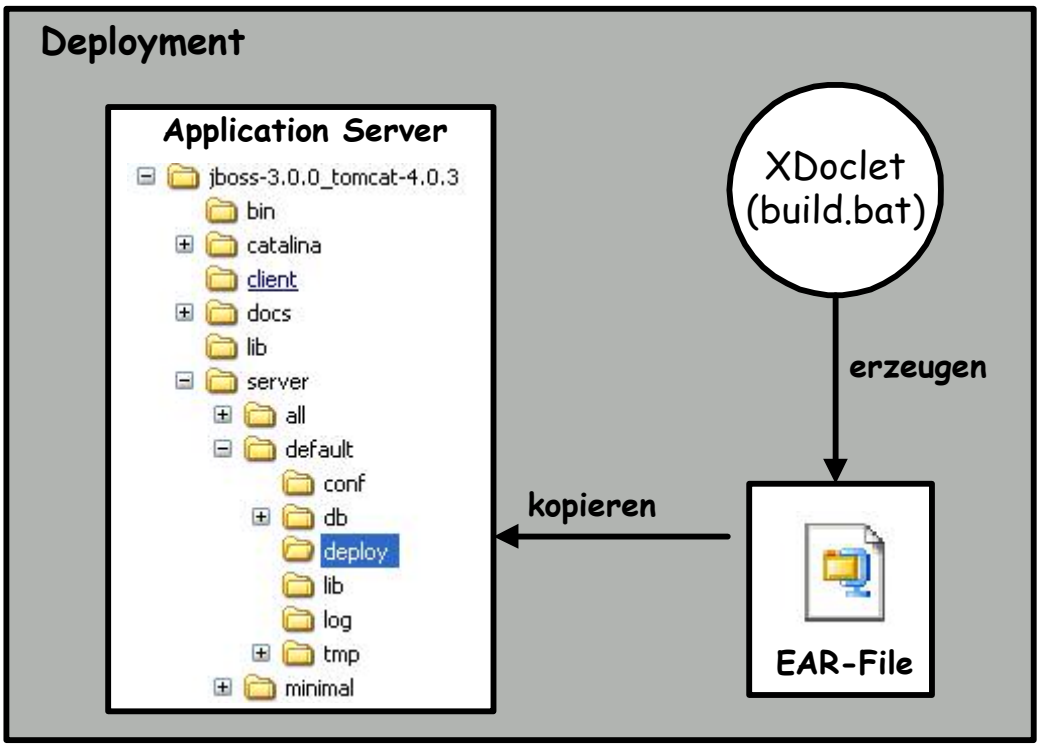

Abbildung 44 - Der Deploy-Prozeß

Der in der Grafik abgebildete JBoss Applikationsserver erkennt automatisch das „Hinzufügen“ oder „Aktualisieren“ einer EAR-Datei. Der Server prüft die Datei und gibt entweder einen Fehler oder einen „positive“ Meldung zurück.

Anhang D - Diagramme

Klassendiagramme

Name	Typ	Funktion
BookmarkBean	EntityBean	Kapselt Datenbankzugriffe für die Bookmarkverwaltung

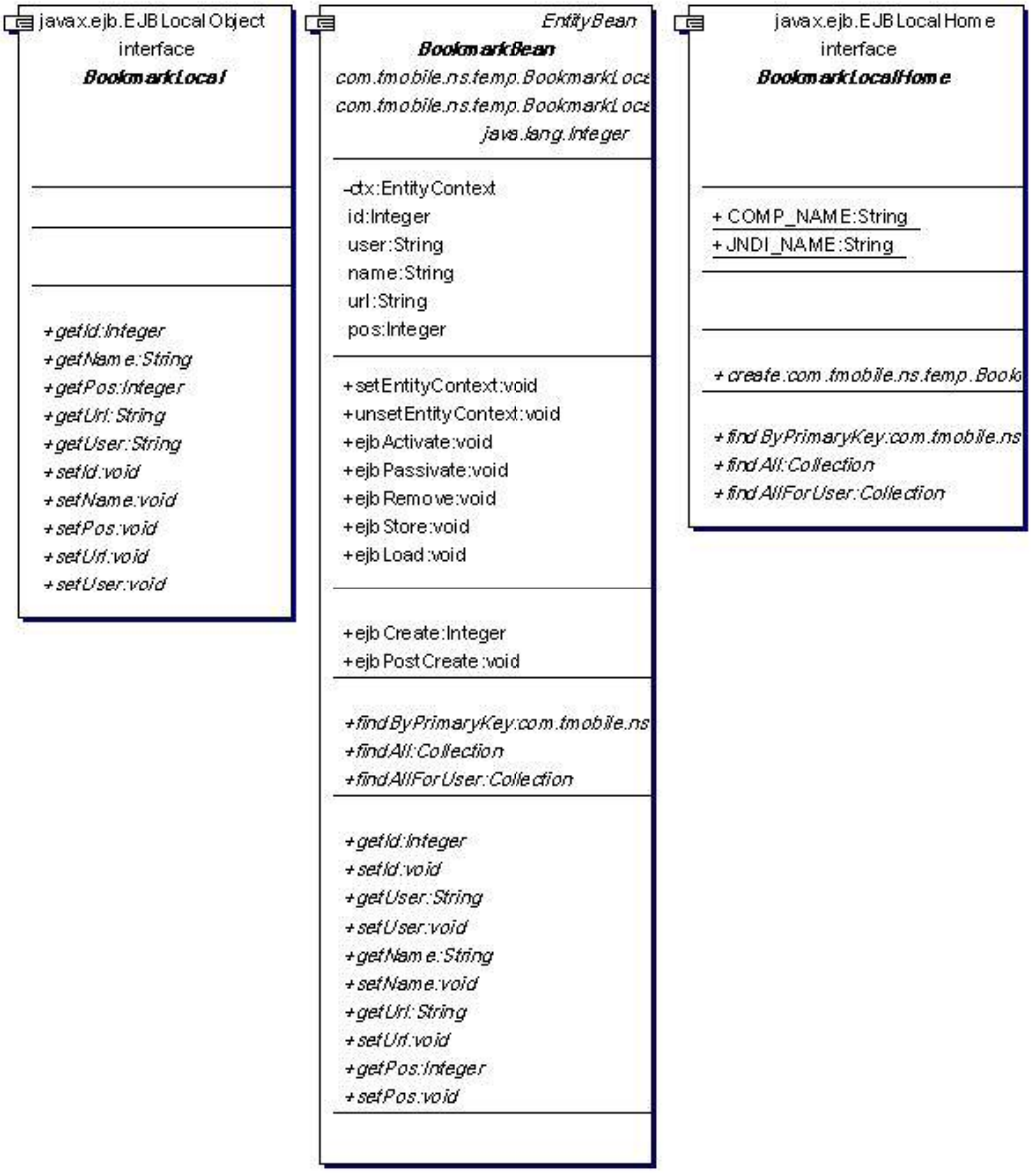

Abbildung 45 - Klassendiagramm BookmarkBean

Name	Typ	Funktion
BookmarkHandlerBean	SessionBean	Stellt dem Steuerungsinterface Funktionen für die Bookmarkverwaltung zur Verfügung

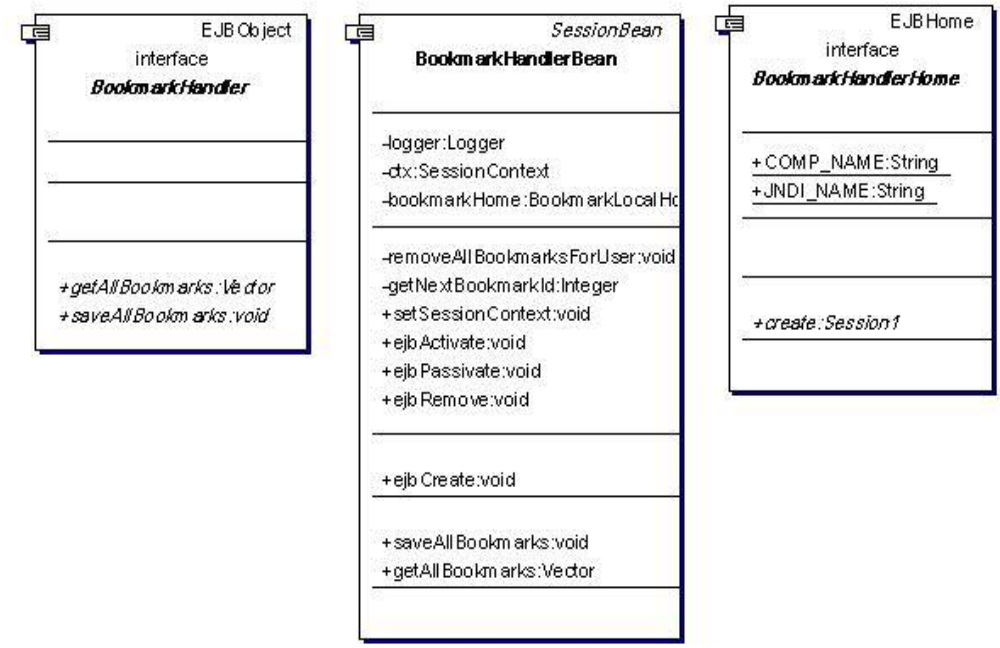

Abbildung 46 - Klassendiagramm BookmarkHandler

Name	Typ	Funktion
CallDataBean	EntityBean	Kapselt Datenbankzugriffe für die Call-Setup Informationen.

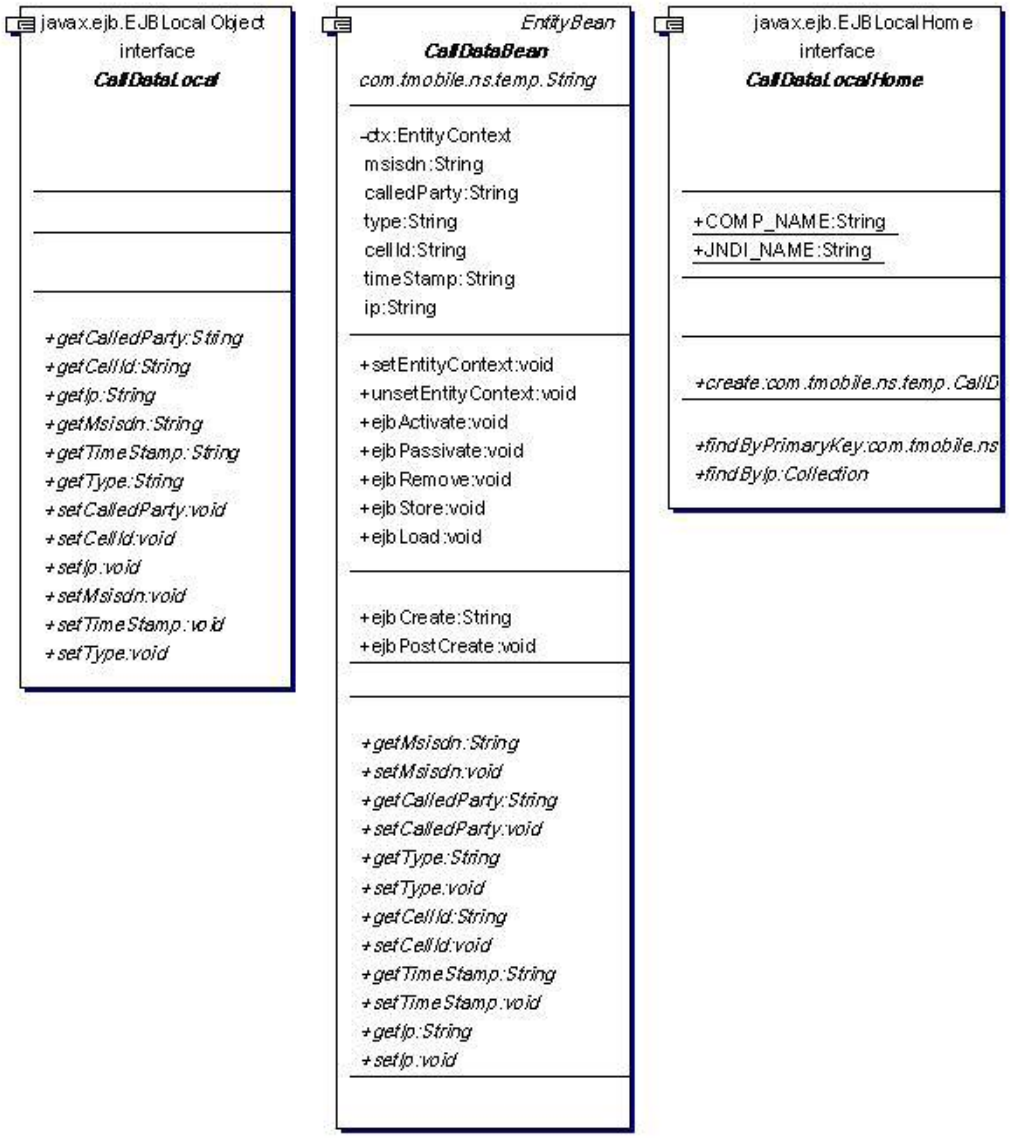

Abbildung 47 - Klassendiagramm CallDataBean

Name	Typ	Funktion
CallHandlerBean	MessageDrivenBean	Führt den Einwahlprozess einer Mobilstation durch und verarbeitet die Call-Setup Informationen.

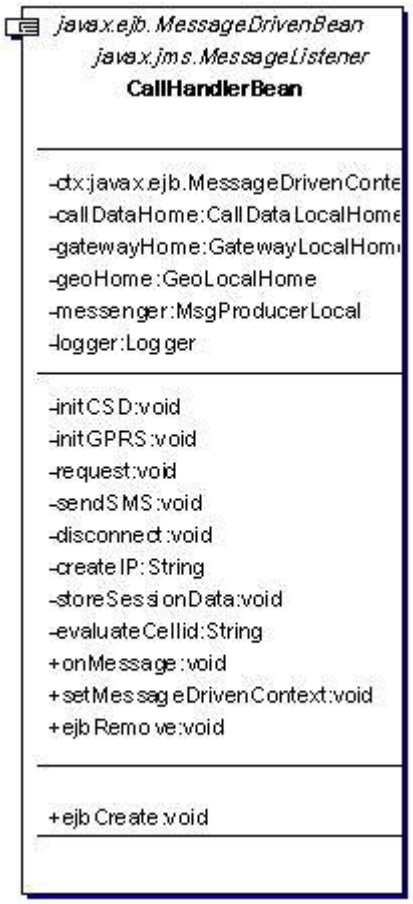

Abbildung 48 - Klassendiagramm CallHandlerBean

Name	Typ	Funktion
Cell	ValueObject	Hält die Daten einer Zelle

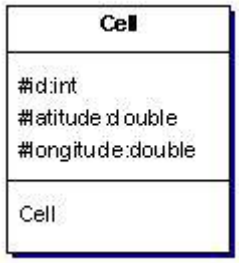

Abbildung 49 - Klassendiagramm Cell

Name	Typ	Funktion
CIABean	SessionBean	Repräsentiert den CIA-Server des Wirknetzes und ermittelt die MSISDN einer Mobilstation anhand der IP-Adresse.

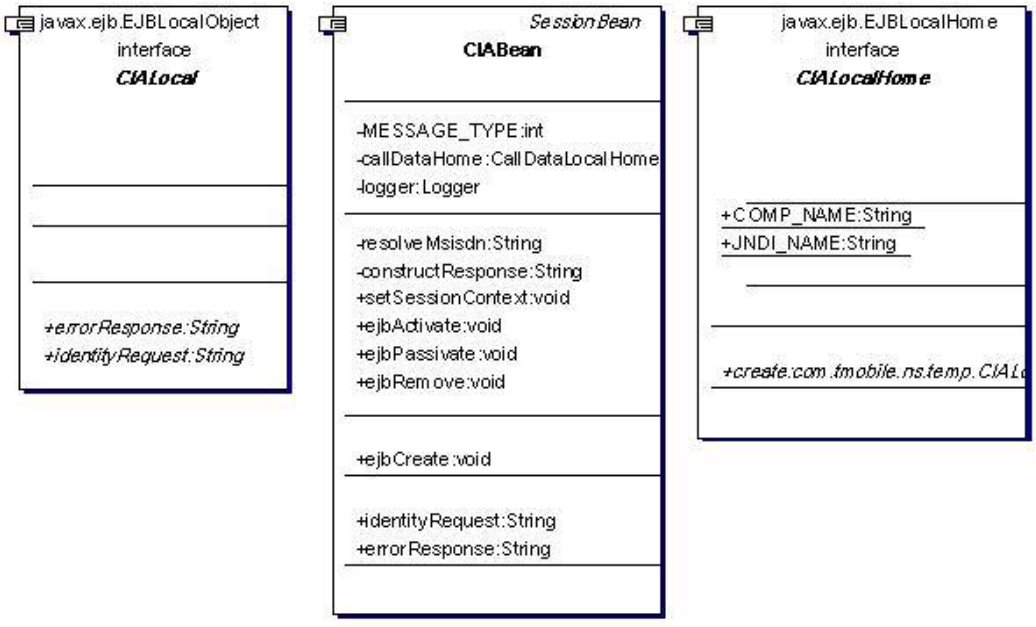

Abbildung 50 - Klassendiagramm CIABean

Name	Typ	Funktion
CrispBean	SessionBean	Repräsentiert den CRISP-Applikations-Server des Wirknetzes und delegiert die VASP-Anfragen an die LBSBean oder CIABean weiter.

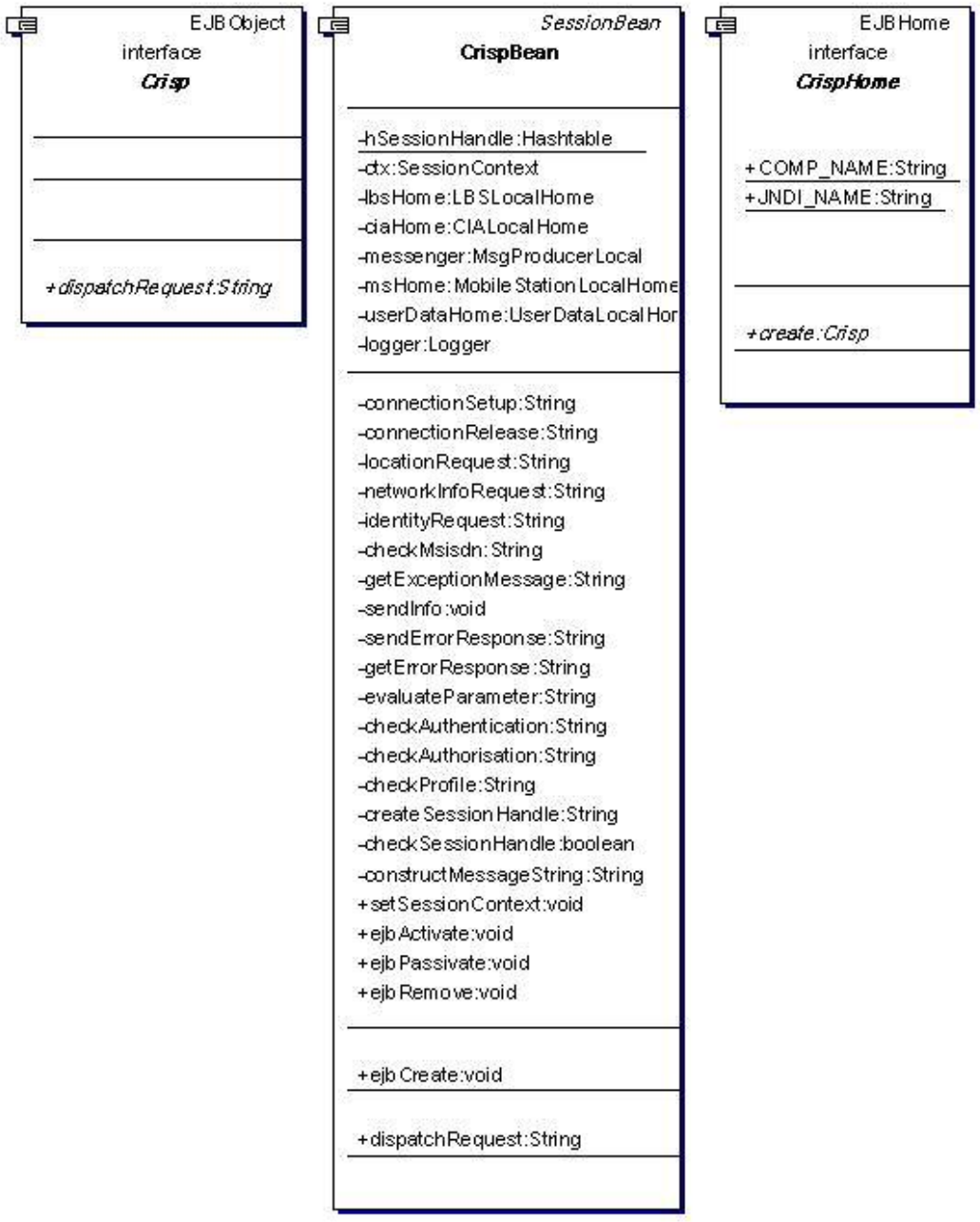

Abbildung 51 - Klassendiagramm CRISP Bean

Name	Typ	Funktion
DataInitializerBean	SessionBean	Speichert beim ersten Start des Steuerungs-interfaces die Zelldaten der Karte und hinterlegt die Standardprofile.

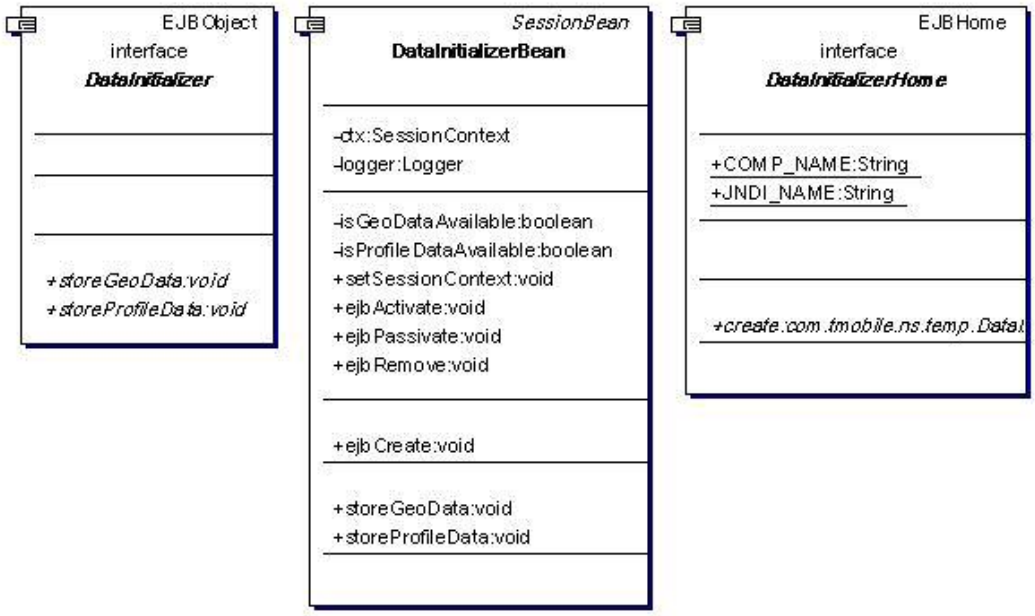

Abbildung 52 - Klassendiagramm DataInitilizerBean

Name	Typ	Funktion
GatewayBean	SessionBean	Nimmt die Internetanfrage (URL) einer Mobilstation entgegen und formuliert daraus einen HTTP-Request. Das Ergebnis wird als Message an die Steuerungsoberfläche bzw. die Mobilstation gesendet.

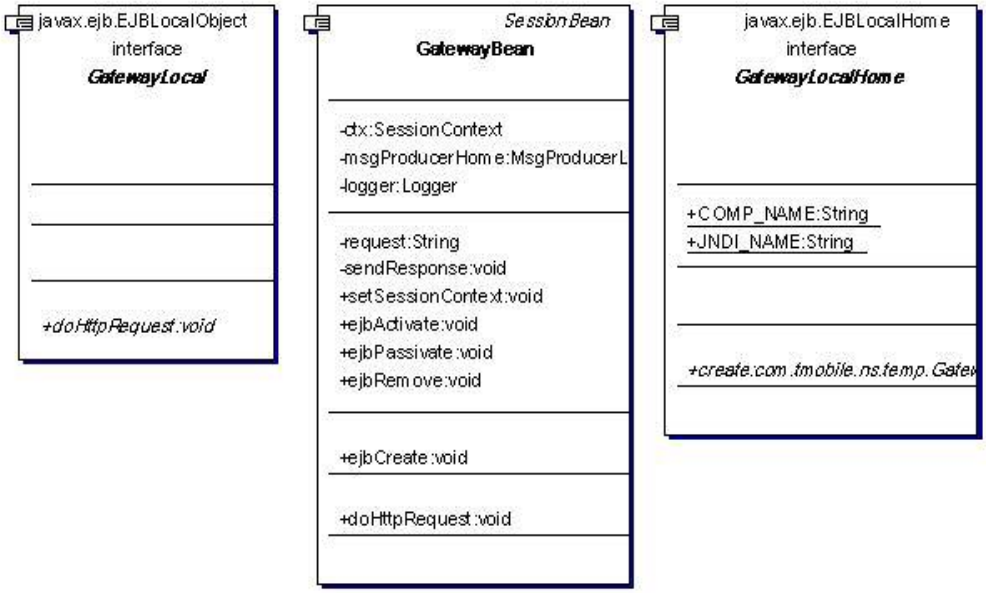

Abbildung 53 - Klassendiagramm GatewayBean

Name	Typ	Funktion
GeoBean	SessionBean	Ermittelt für ein Koordinatenpaar die Zell-ID

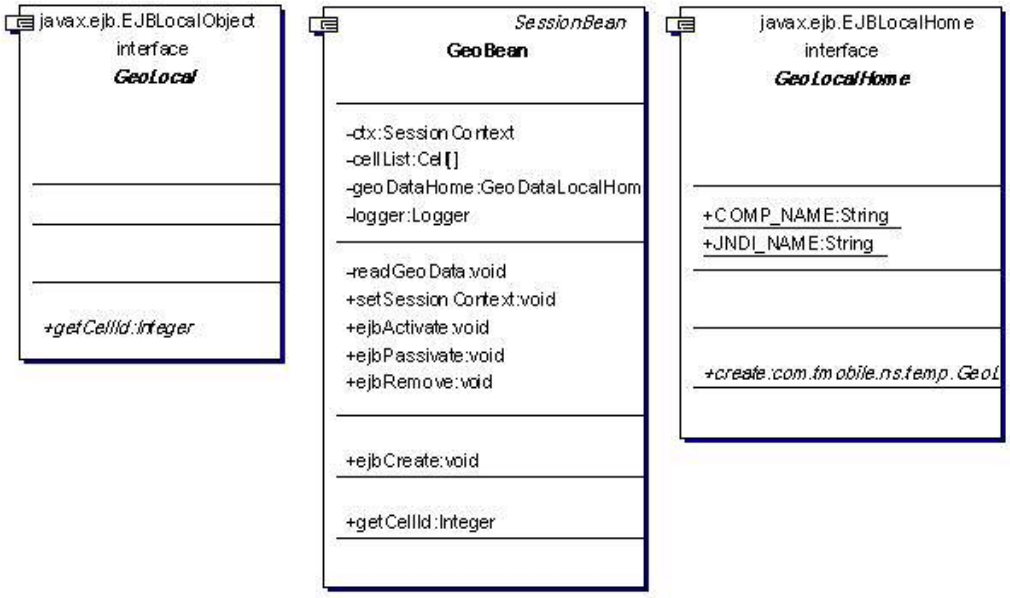

Abbildung 54 - Klassendiagramm GeoBean

Name	Typ	Funktion
GeoDataBean	EntityBean	Kapselt die Datenbankzugiffe für die Geodaten. Berechnet die Position einer Mobilstation in unterschiedlichen Formaten.

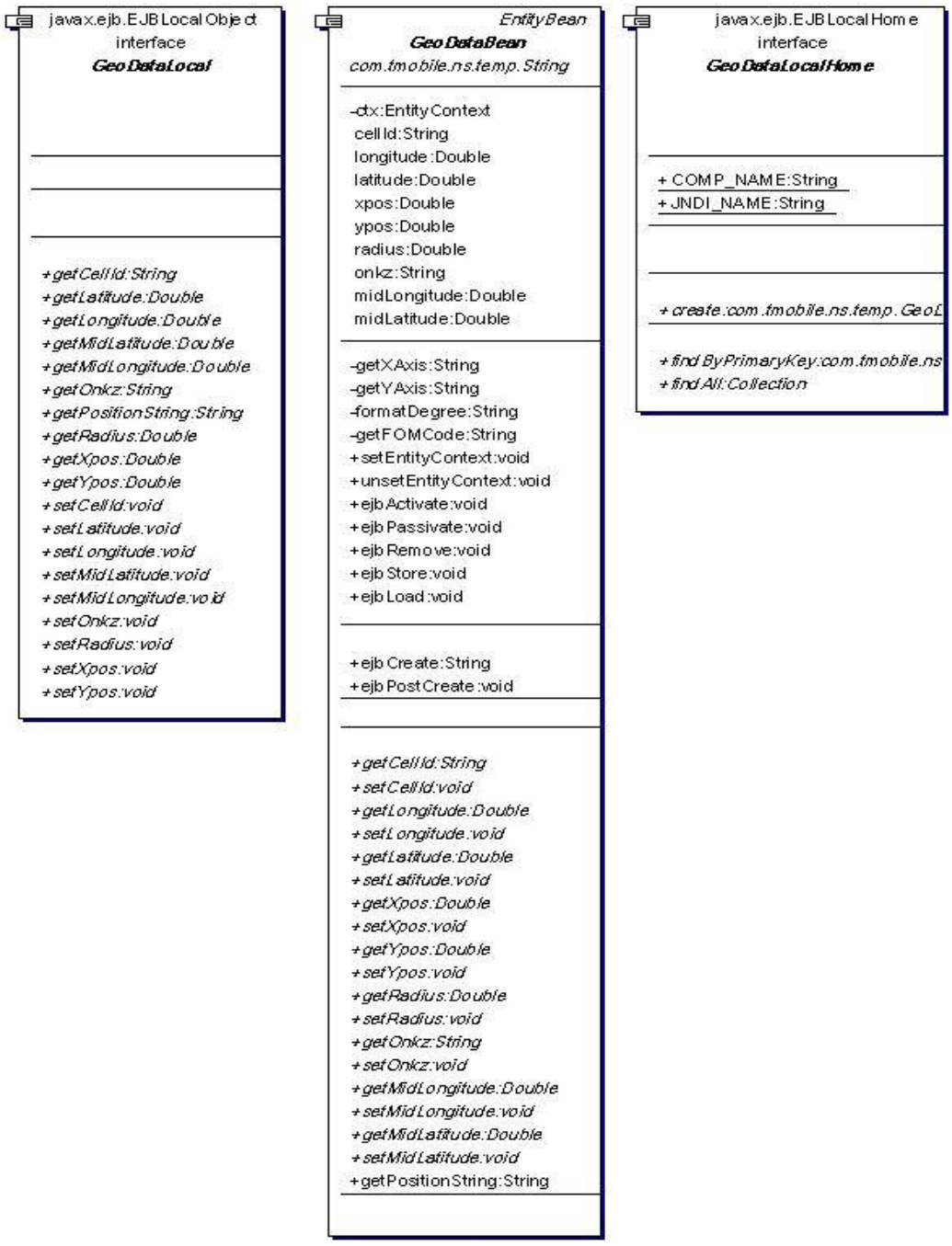

Abbildung 55 - Klassendiagramm GeoDataBean

Name	Typ	Funktion
LBSBean	EntityBean	Repräsentiert den LBS-Server des Wirknetzes. Implementiert den Location Request und stellt im Rahmen des Dienstes die Positionsdaten einer Mobilstation bereit. Implementiert den NetworkInfo Request, der zur einer Zell-ID die Koordinaten ermittelt.

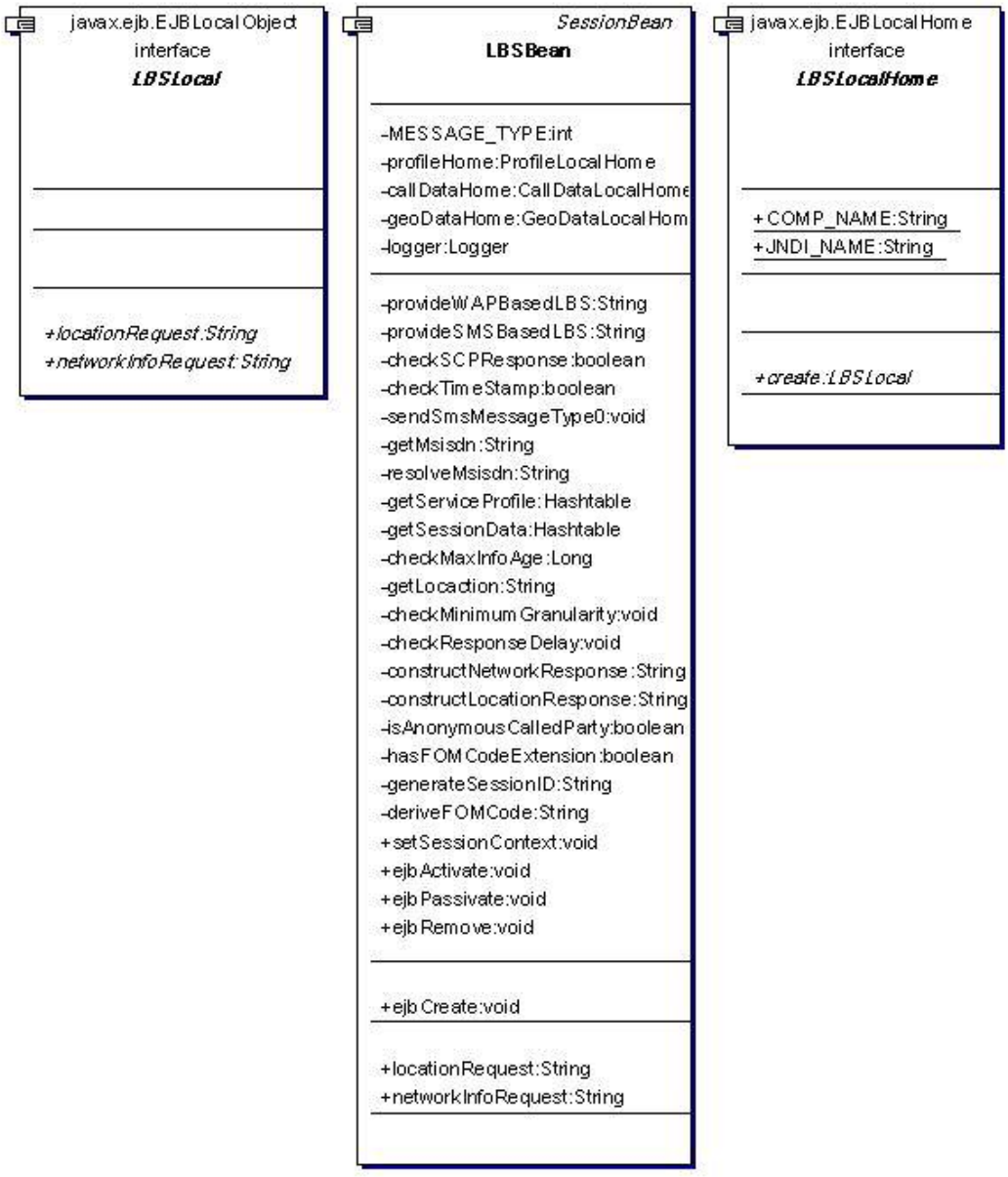

Abbildung 56 - Klassendiagramm LBSBean

Name	Typ	Funktion
LBSException	Exception	Wird von der LBSBean erzeugt, falls ein Fehler auftritt. Dabei handelt es sich gewöhnlich um einen ungültigen Parameter.

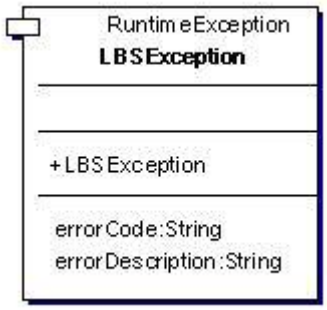

Abbildung 57 - Klassendiagramm LBSException

Name	Typ	Funktion
MobileHandlerBean	SessionBean	Stellt der Steuerungsoberfläche Funktionen zur Verwaltung der Mobilstation zur Verfügung

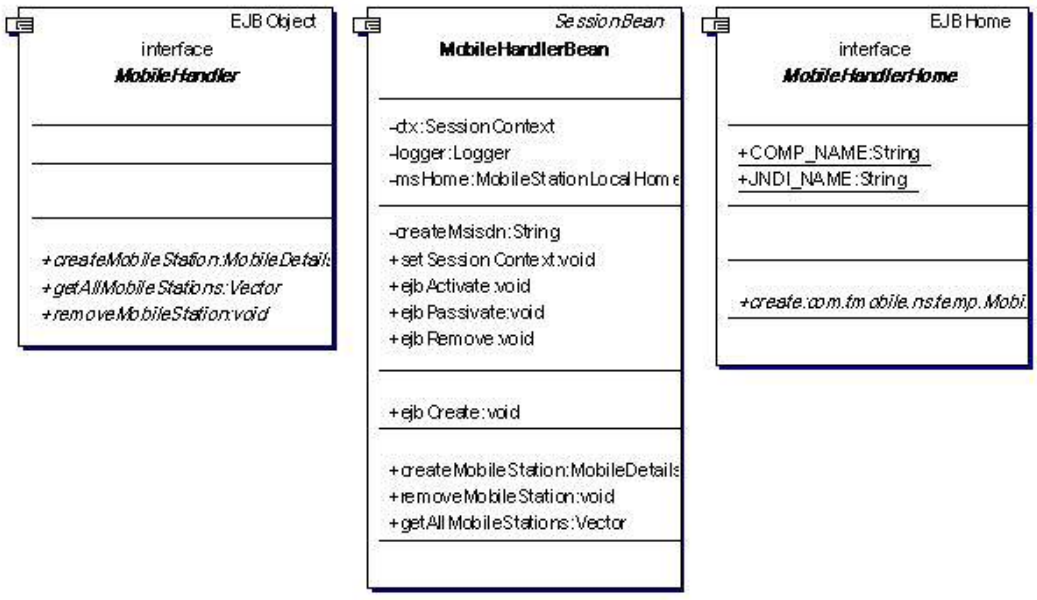

Abbildung 58 - Klassendiagramm MobileHandlerBean

Name	Typ	Funktion
MobileStation	EntityBean	Kapselt die Datenbankzugriffe für die Daten einer Mobilstation.

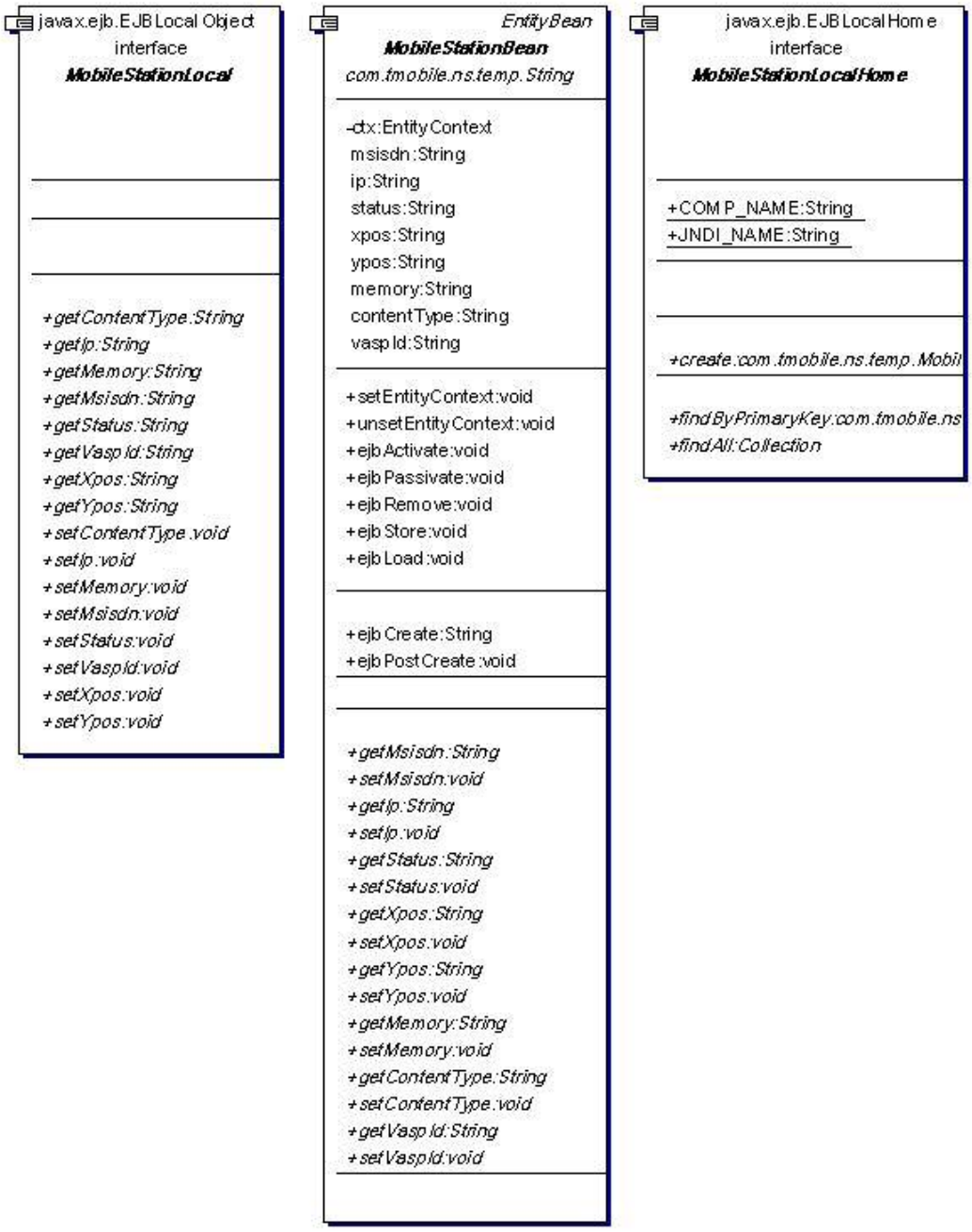

Abbildung 59 - Klassendiagramm MobileStationBean

Name	Typ	Funktion
MsgProducerBean	SessionBean	Erzeugt Nachrichten und sendet diese an ein Topic.

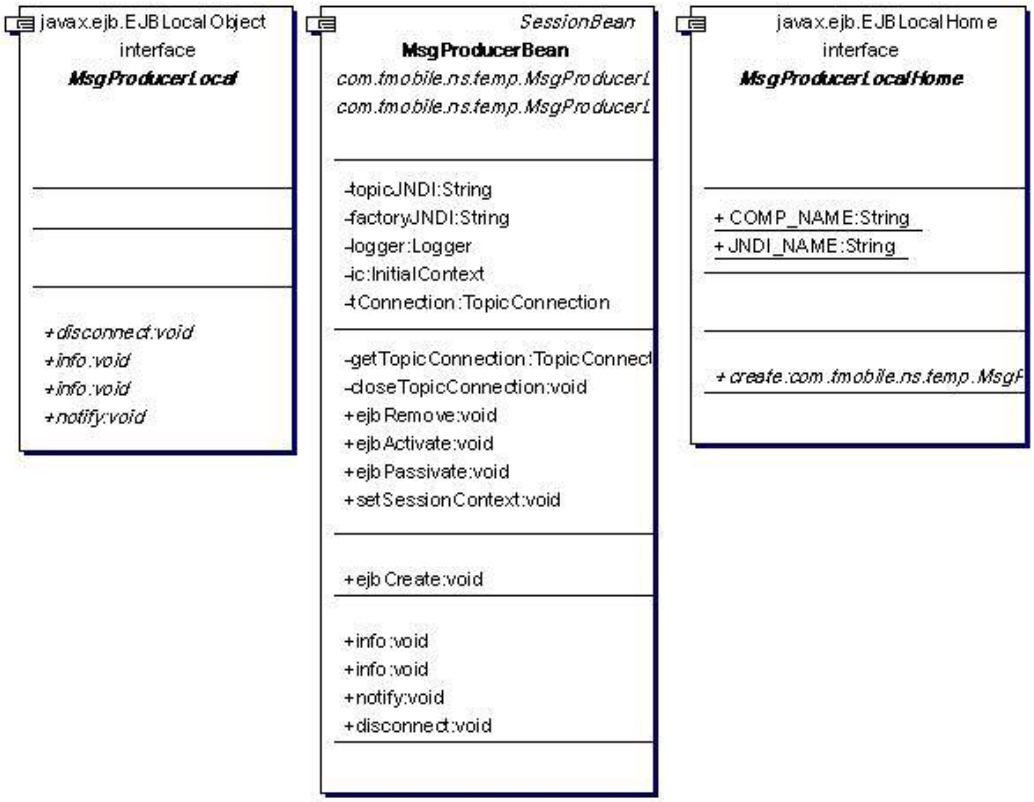

Abbildung 60 - Klassendiagramm MsgProducerBean

Name	Typ	Funktion
ProfileBean	EntityBean	Kapselt die Datenbankzugriffe für die Profildaten.

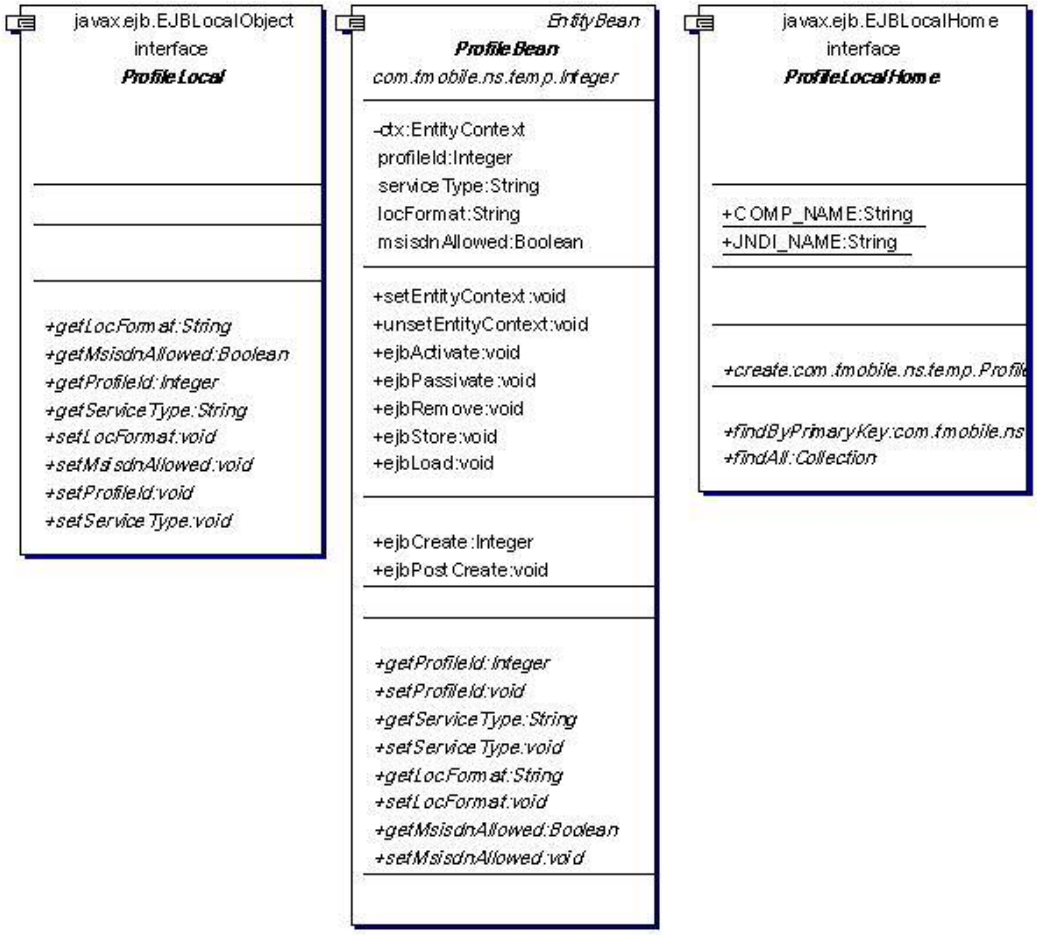

Abbildung 61 - Klassendiagramm ProfileBean

Name	Typ	Funktion
ProfileHandlerBean	SessionBean	Stellt der Steuerungsoberfläche Funktionen zur Verwaltung der Profile zur Verfügung.

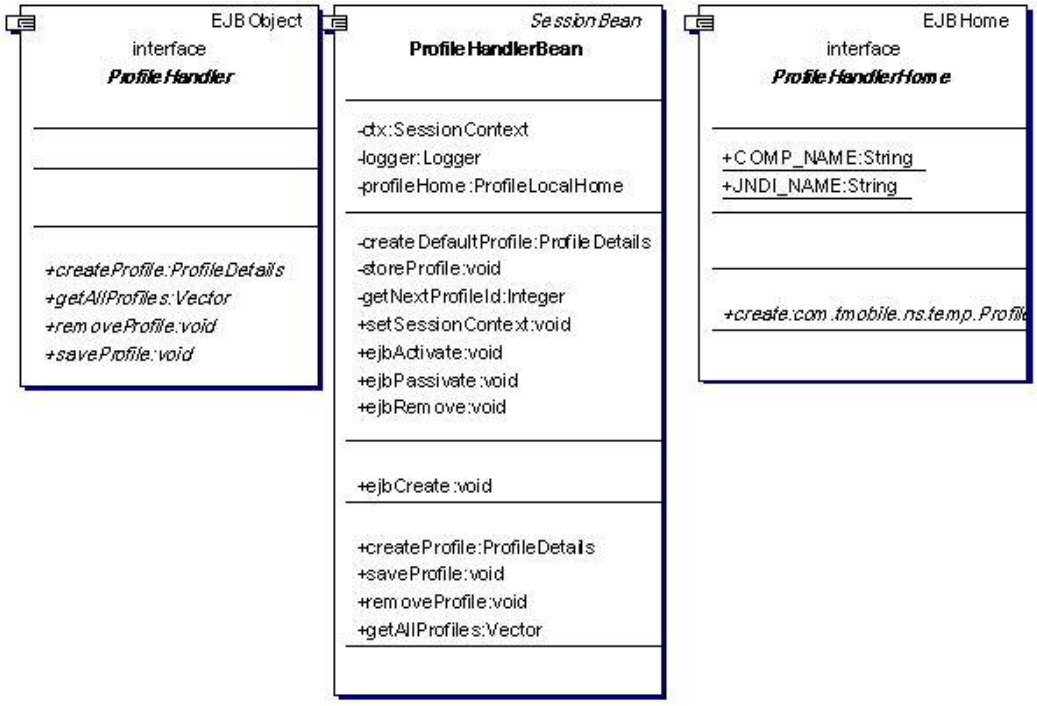

Abbildung 62 - Klassendiagramm ProfileHandlerBean

Name	Typ	Funktion
UserDataBean	EntityBean	Kapselt die Datenbankzugriffe für die Userdaten. *User* und *Passwort* werden für den Login in der Steuerungsoberfläche benötigt. Die *VASP-ID* wird bei der Anfrage eines VASPs übergeben.

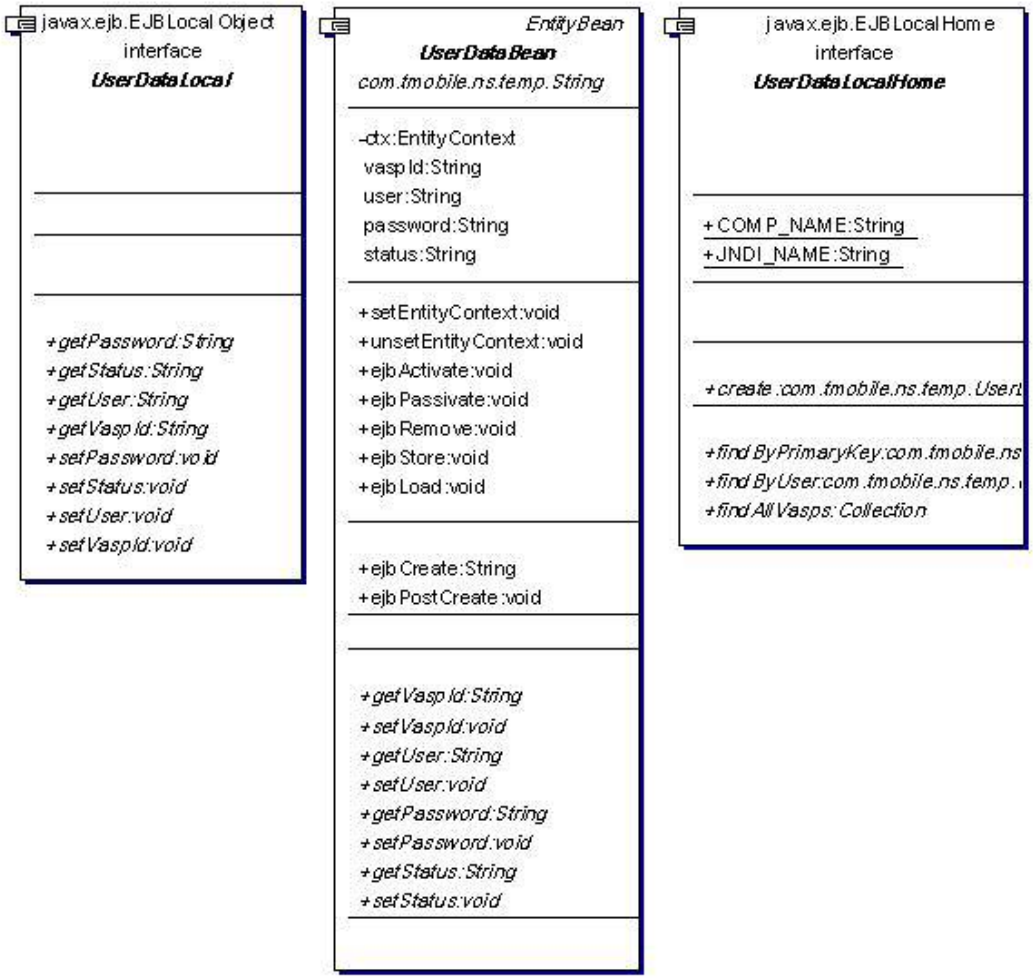

Abbildung 63 - Klassendiagramm UserDataBean

Name	Typ	Funktion
UserHandlerBean	SessionBean	Stellt der Steuerungsoberfläche Funktionen zur Verwaltung der Userdaten zur Verfügung.

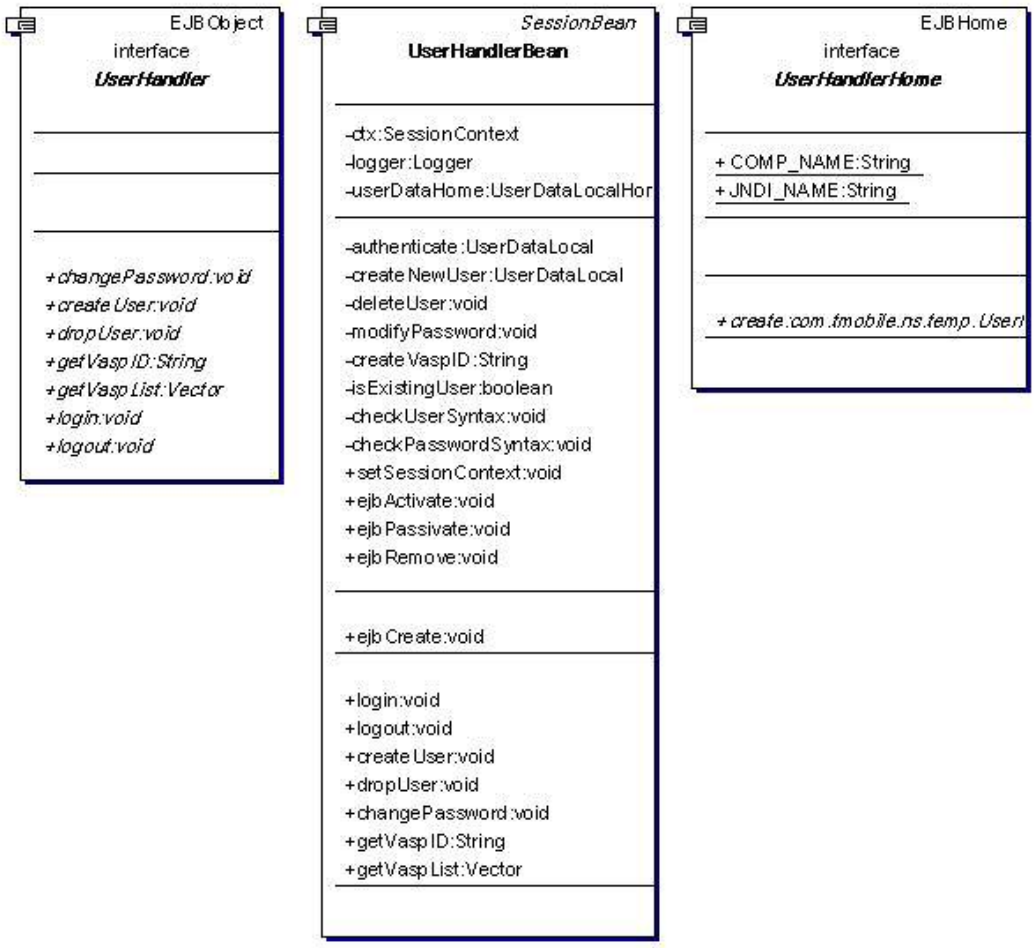

Abbildung 64 - Klassendiagramm UserHandlerBean

Sequenzdiagramme

❖ Location Request

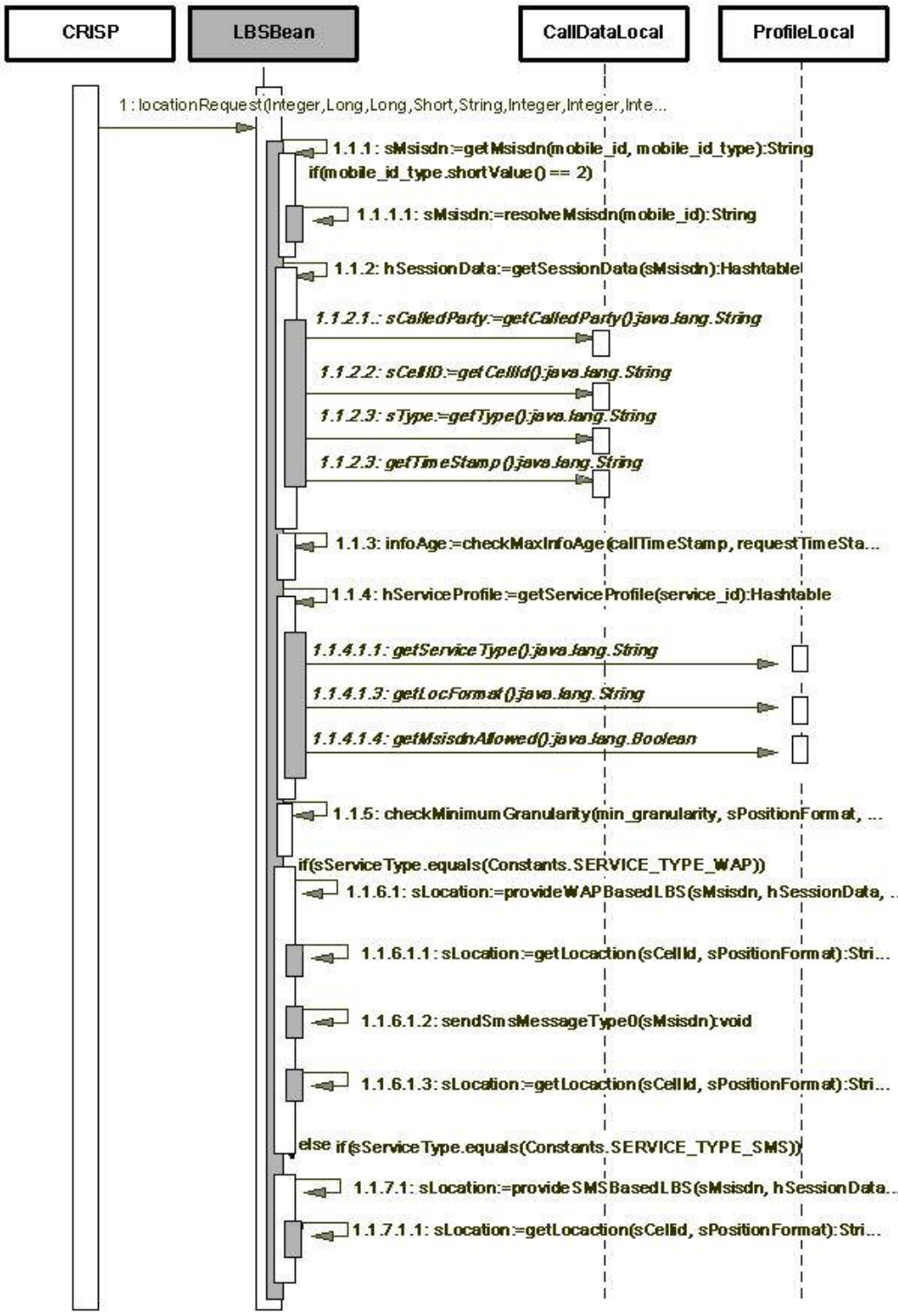

Abbildung 65 - Sequenzdiagramm Location Request

- **Network Info Request**

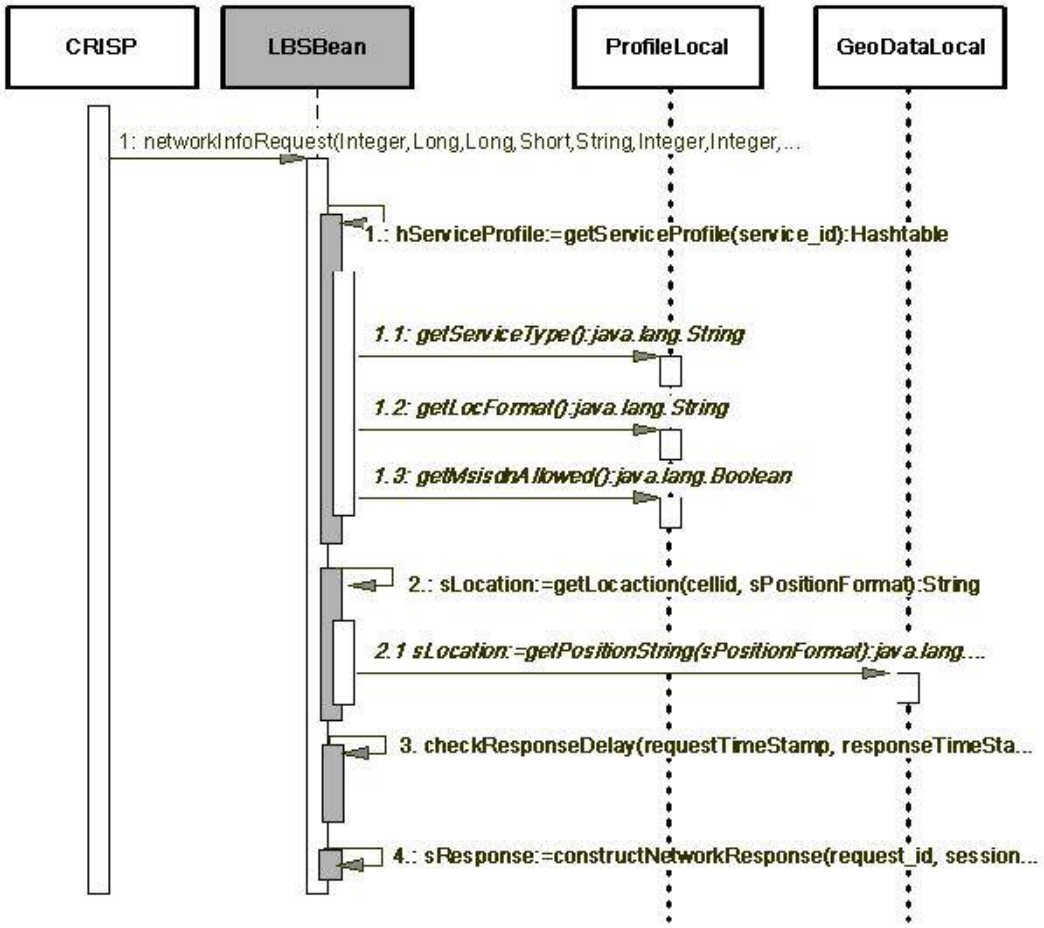

Abbildung 66 - Sequenzdiagramm NetworkInfo Request

- **Identity Request**

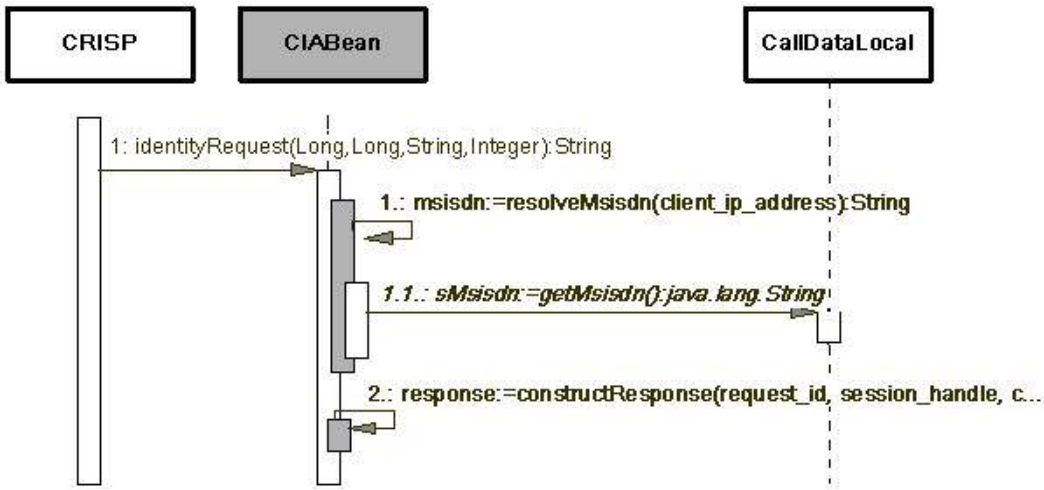

Abbildung 67 - Sequenzdiagramm Identity Request

Ich versichere die von mir vorgelegte Arbeit selbständig verfasst zu haben. Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder nicht veröffentlichten Arbeiten anderer entnommen sind, habe ich als entnommen kenntlich gemacht. Sämtliche Quellen und Hilfsmittel, die ich für die Arbeit benutzt habe, sind angegeben. Die Arbeit hat mit gleichem Inhalt bzw. in wesentlichen Teilen noch keiner anderen Prüfungsbehörde vorgelegen.

Frechen, den 24. Januar 2003

Ramin Ziai